

OF COMMERCE
MARK OFFICE
231
RETURN IN TEN DAYS

AN EQUAL OPPORTUNITY EMPLO



U.S. OFFICIAL MAIL	
U.S. POSTAGE	
PENALTY FOR PRIVATE USE \$300	03.95
METER H 550900	

☐ OTHER
TRESSED



VER 131

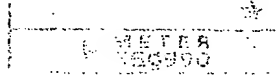
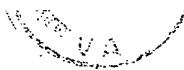
H/V

MARK OFFICE

231

RETURN IN TEN DAYS

AN EQUAL OPPORTUNITY EMPLOYER



☐ OTHER

POSTPAID



4/2

Organization TC2100 Bldg./Room PK2
U. S. DEPARTMENT OF COMMERCE
PATENT AND TRADEMARK OFFICE
WASHINGTON, DC 20231
IF UNDELIVERABLE RETURN IN TEN DAYS

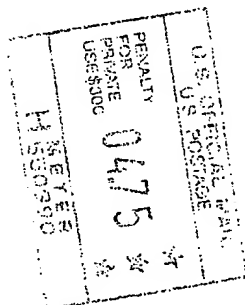
OFFICIAL BUSINESS

AN EQUAL OPPORTUNITY EMPLOYER

- ☐ A INSUFFICIENT ADDRESS
☐ C ATTEMPTED NOT KNOWN
☐ S NO SUCH NUMBER/ STREET
☒ NOT DELIVERABLE AS ADDRESSED
UNABLE TO FORWARD

☐ OTHER

RTS
RETURN TO SENDER



DELIVERED

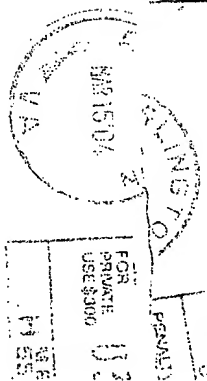
3/4

PATENT AND TRADEMARK OFFICE
WASHINGTON, DC 20231
IF UNDELIVERABLE RETURN IN TEN DAYS

OFFICIAL BUSINESS

AN EQUAL OPPORTUNITY EMPLOYER

- ☐ A ☐ INSUFFICIENT ADDRESS
☐ C ☐ ATTEMPTED NOT KNOWN ☐ OTHER
☒ S ☐ NO SUCH NUMBER/STREET
☒ NOT DELIVERABLE AS ADDRESSED
- UNABLE TO FORWARD



4/4



UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
09/668,995	09/25/2000	Katsue Kojima	FUJY17.788	3410

7590

03/15/2004

Helgott & Karas PC
Empire State Building
60th Floor
New York, NY 10118

EXAMINER

WASSUM, LUKE S

ART UNIT

PAPER NUMBER

2177

DATE MAILED: 03/15/2004

Please find below and/or attached an Office communication concerning this application or proceeding.

RECEIVED

MAR 26 2004

Technology Center 2100

Office Action Summary

Application No.

09/668,995

Applicant(s)

KOJIMA ET AL.

Examiner

Luke S. Wassum

Art Unit

2177

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTH(S) FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If the period for reply specified above is less than thirty (30) days, a reply within the statutory minimum of thirty (30) days will be considered timely.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133).
- Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

- 1) ☒ Responsive to communication(s) filed on 25 September 2000.
- 2a) ☐ This action is **FINAL**. 2b) ☒ This action is non-final.
- 3) ☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

- 4) ☒ Claim(s) 1-23 is/are pending in the application.
- 4a) Of the above claim(s) _____ is/are withdrawn from consideration.
- 5) ☐ Claim(s) _____ is/are allowed.
- 6) ☒ Claim(s) 1-23 is/are rejected.
- 7) ☐ Claim(s) _____ is/are objected to.
- 8) ☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

- 9) ☒ The specification is objected to by the Examiner.
- 10) ☒ The drawing(s) filed on 25 September 2000 is/are: a) ☒ accepted or b) ☐ objected to by the Examiner.
- Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).
- Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).
- 11) ☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. §§ 119 and 120

- 12) ☒ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).
- a) ☒ All b) ☐ Some * c) ☐ None of:
1. ☒ Certified copies of the priority documents have been received.
 2. ☐ Certified copies of the priority documents have been received in Application No. _____.
 3. ☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).
- * See the attached detailed Office action for a list of the certified copies not received.
- 13) ☐ Acknowledgment is made of a claim for domestic priority under 35 U.S.C. § 119(e) (to a provisional application) since a specific reference was included in the first sentence of the specification or in an Application Data Sheet. 37 CFR 1.78.
- a) ☐ The translation of the foreign language provisional application has been received.
- 14) ☐ Acknowledgment is made of a claim for domestic priority under 35 U.S.C. §§ 120 and/or 121 since a specific reference was included in the first sentence of the specification or in an Application Data Sheet. 37 CFR 1.78.

Attachment(s)

- 1) ☒ Notice of References Cited (PTO-892)
- 2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)
- 3) ☒ Information Disclosure Statement(s) (PTO-1449) Paper No(s) 2.
- 4) ☐ Interview Summary (PTO-413) Paper No(s). _____
- 5) ☐ Notice of Informal Patent Application (PTO-152)
- 6) ☐ Other: _____

DETAILED ACTION

Priority

1. Receipt is acknowledged of papers submitted under 35 U.S.C. 119(a)-(d), which papers have been placed of record in the file.

A priority date of 17 November 1999, based on Japanese patent document 11-327483, has been established for the instant application.

Information Disclosure Statement

2. The Applicants' Information Disclosure Statement, filed 25 September 2000, has been received and entered into the record. Since the Information Disclosure Statement conforms to the provisions of MPEP § 609, the documents referenced therein have been considered. The examiner notes that only the English language abstracts of the foreign language references have been considered. See attached form PTO-1449.

Corrected Filing Receipt

3. The examiner acknowledges receipt of the Applicants' request for a corrected filing receipt. The requested correction (inclusion of foreign priority information) has been made.

Specification

4. The disclosure is objected to because of the following informalities:

Art Unit: 2177

The Brief Description of the Drawings is objected to, because it fails to describe each drawing in specific enough terms. The description of each drawing should be detailed enough to distinguish one drawing from another. See MPEP § 608.01(f) and 37 C.F.R. § 1.74.

There is a typographical error on page 10, line 12: "input side line handler 3" should be "input side line handler 2".

Appropriate correction is required.

5. The disclosure is objected to because of the following informalities:

In the Brief Description of the Drawings, the examiner notes that many drawing figures are described using identical broad descriptions. The Applicants are required to provide more specific descriptions such that each drawing has a distinct and clear description.

Appropriate correction is required.

Claim Objections

6. Claim 10 is objected to because of the following informalities:

The claim makes reference to a 'minimum address of a continuous empty area'. The examiner assumes that what is meant is the 'starting address'.

Appropriate correction is required.

Claim Rejections - 35 USC § 112

7. The following is a quotation of the second paragraph of 35 U.S.C. 112:

The specification shall conclude with one or more claims particularly pointing out and distinctly claiming the subject matter which the applicant regards as his invention.

8. Claims 4, 9, 10 and 12-14 are rejected under 35 U.S.C. 112, second paragraph, as being indefinite for failing to particularly point out and distinctly claim the subject matter which applicant regards as the invention.

9. Regarding claim 4, the limitation "...invalidity informer informing the invalidation of the pointer..." renders the claim indefinite, because it fails to disclose what/who is being informed.

10. Regarding claim 9, the limitation "...biggest address..." renders the claim indefinite, because addresses are pointers to data, and do not have the claimed characteristic of 'bigness'.

11. Regarding claim 10, the limitation "...each by the size of the continuous empty area..." renders the claim indefinite, because it is unclear what this limitation is meant to apply to, nor what the limitation is claiming.

12. Regarding claim 12, the limitation "...link information related to a link between data setting areas..." renders the claim indefinite, because the limitation fails to recite what precisely the link information is, but merely that it is 'related' to other data.

13. Regarding claim 13, the limitation "...information related to the use-condition ..." renders the claim indefinite, because the limitation fails to recite what precisely the information is, but merely that it is 'related' to other data.

Furthermore, the limitation "...information of the data setting area of a destination ..." renders the claim indefinite, because it is unclear what exactly the information is.

14. Regarding claim 14, the limitation "...on the basis of the frequency data ..." renders the claim indefinite, because the limitation fails to recite how the data setting area is chosen, and precisely what effect the frequency data has on this determination.

Claim Rejections - 35 USC § 102

15. The following is a quotation of the appropriate paragraphs of 35 U.S.C. 102 that form the basis for the rejections under this section made in this Office action:

A person shall be entitled to a patent unless –

(b) the invention was patented or described in a printed publication in this or a foreign country or in public use or on sale in this country, more than one year prior to the date of application for patent in the United States.

16. Claims 15, 19 and 23 are rejected under 35 U.S.C. 102(b) as being anticipated by **Hitz et al.** (U.S. Patent 5,819,292).

17. Regarding claim 15, **Hitz et al.** teaches a data management apparatus for managing a plurality of data which are used in order to execute an application program for providing services related to communication by a switching system, comprising:

- a) a data field storing the plurality of data, said data field is composed of a plurality of data setting areas, each data in said data field is stored by a single of plural data setting areas according to a size of data (see details of the inode structure, and how larger files are stored across multiple blocks and possibly multiple inodes, col. 5, line 60 through col. 6, line 52; see also Figures 4A through 4D);

- b) a data setting area management table storing information related to the use-condition of each data setting area (see disclosure of the block map and inode file, both of which provide analogous functionality, col. 9, line 50 through col. 10, line 48);
- c) an allocation controller referring to said data setting area management table, and determining at least one of empty data setting areas in order to allocate a data requested to be added (see disclosure that new data is only written to unallocated blocks, col. 4, lines 14-16); and
- d) an adder storing the data requested to be added to at least one of the empty data setting areas which is determined by said allocation controller (see disclosure that new data is only written to unallocated blocks, col. 4, lines 14-16).

18. Regarding claim 19, **Hitz et al.** teaches a method for managing a plurality of data which are used in order to execute an application program for providing services related to communication by a switching system, comprising steps of:

- a) storing the plurality of data into a data field, said data field is composed of a plurality of data setting areas, each data in said data field is stored by a single of plural data setting areas according to a size of data (see details of the inode structure, and how larger files are stored across multiple blocks and possibly multiple inodes, col. 5, line 60 through col. 6, line 52; see also Figures 4A through 4D);
- b) storing information related to the use-condition of each data setting area into a data setting area management table (see disclosure of the block map and inode file, both of which provide analogous functionality, col. 9, line 50 through col. 10, line 48);

- c) referring to said data setting area management table, and determining at least one of empty data setting areas in order to allocate a data requested to be added (see disclosure that new data is only written to unallocated blocks, col. 4, lines 14-16); and
- d) storing the data requested to be added to at least one of the empty data setting areas which is determined (see disclosure that new data is only written to unallocated blocks, col. 4, lines 14-16).

19. Regarding claim 23, **Hitz et al.** teaches a computer readable medium storing a program for managing a plurality of data which are used in order to execute an application program for providing services related to communication by a switching system, the program comprising steps of:

- a) storing the plurality of data into a data field, said data field is composed of a plurality of data setting areas, each data in said data field is stored by a single of plural data setting areas according to a size of data (see details of the inode structure, and how larger files are stored across multiple blocks and possibly multiple inodes, col. 5, line 60 through col. 6, line 52; see also Figures 4A through 4D);
- b) storing information related to the use-condition of each data setting area in a data setting area management table (see disclosure of the block map and inode file, both of which provide analogous functionality, col. 9, line 50 through col. 10, line 48);
- c) referring to said data setting area management table, and determining at least one of empty data setting areas in order to allocate a data requested to be added (see disclosure that new data is only written to unallocated blocks, col. 4, lines 14-16); and

- d) storing the data requested to be added to at least one of the empty data setting areas which is determined (see disclosure that new data is only written to unallocated blocks, col. 4, lines 14-16).

Claim Rejections - 35 USC § 103

20. The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

(a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negated by the manner in which the invention was made.

21. The factual inquiries set forth in *Graham v. John Deere Co.*, 383 U.S. 1, 148 USPQ 459 (1966), that are applied for establishing a background for determining obviousness under 35 U.S.C. 103(a) are summarized as follows:

1. Determining the scope and contents of the prior art.
2. Ascertaining the differences between the prior art and the claims at issue.
3. Resolving the level of ordinary skill in the pertinent art.
4. Considering objective evidence present in the application indicating obviousness or nonobviousness.

22. This application currently names joint inventors. In considering patentability of the claims under 35 U.S.C. 103(a), the examiner presumes that the subject matter of the various claims was commonly owned at the time any inventions covered therein were made absent any evidence to the contrary. Applicant is advised of the obligation under 37 CFR 1.56 to point out the inventor and invention dates of each claim that was not commonly owned at the time a later invention was made

Art Unit: 2177

in order for the examiner to consider the applicability of 35 U.S.C. 103(c) and potential 35 U.S.C. 102(e), (f) or (g) prior art under 35 U.S.C. 103(a).

23. Claims 1, 2, 16, 17, 20 and 21 are rejected under 35 U.S.C. 103(a) as being unpatentable over **Admitted Prior Art** in view of **Koyama** (Japanese Patent Publication 7-295814).

24. Regarding claim 1, the Applicants teach as **Admitted Prior Art** a data management apparatus for managing a plurality of data which are used in order to execute an application program for providing services related to communication by a switching system substantially as claimed, comprising:

- a) a data field storing the plurality of data (see data field in Figure 21, labeled as Prior Art; see also Applicants' specification, page 3, last paragraph through page 4, first paragraph);
- b) an address acquirer acquiring an address of the data in said data field for which an access is requested by the application program (see pointer acquisition step in Figure 21, labeled as Prior Art; see also Applicants' specification, page 3, last paragraph through page 4, first paragraph); and
- c) a lender lending the address of the data to the application program (see pointer lending step in Figure 21, labeled as Prior Art; see also Applicants' specification, page 3, last paragraph through page 4, first paragraph).

The **Admitted Prior Art** fails to explicitly teach a data management apparatus comprising a lending pointer table storing at least one of pointer records having the acquired address and a

Art Unit: 2177

pointer corresponding to the acquired address, and a lender reading out the pointer from the lending pointer table.

Koyama, however, teaches a data management apparatus comprising a lending pointer table storing at least one of pointer records having the acquired address and a pointer corresponding to the acquired address, and a lender reading out the pointer from the lending pointer table (see Constitution section, page 2; see also text of claim 1, beginning on page 2; see also paragraph [0016], beginning on page 10).

It would have been obvious to one of ordinary skill in the art at the time of the invention to incorporate a lending pointer table for storing pointer records to the acquired address, since this would conceal the details of the address management mechanism from the application program, thus precluding a specific application from accessing data improperly (see text of claim 2, page 3).

25. Regarding claim 16, the Applicants teach as **Admitted Prior Art** a method for managing a plurality of data which are used in order to execute an application program for providing services related to communication by a switching system substantially as claimed, comprising:

- a) a data field storing the plurality of data (see data field in Figure 21, labeled as Prior Art; see also Applicants' specification, page 3, last paragraph through page 4, first paragraph);
- b) an address acquirer acquiring an address of the data in said data field for which an access is requested by the application program (see pointer acquisition step in Figure 21, labeled as Prior Art; see also Applicants' specification, page 3, last paragraph through page 4, first paragraph); and

- c) a lender lending the address of the data to the application program (see pointer lending step in Figure 21, labeled as Prior Art; see also Applicants' specification, page 3, last paragraph through page 4, first paragraph).

The **Admitted Prior Art** fails to explicitly teach a method comprising a lending pointer table storing at least one of pointer records having the acquired address and a pointer corresponding to the acquired address, and a lender reading out the pointer from the lending pointer table.

Koyama, however, teaches a method comprising a lending pointer table storing at least one of pointer records having the acquired address and a pointer corresponding to the acquired address, and a lender reading out the pointer from the lending pointer table (see Constitution section, page 2; see also text of claim 1, beginning on page 2; see also paragraph [0016], beginning on page 10).

It would have been obvious to one of ordinary skill in the art at the time of the invention to incorporate a lending pointer table for storing pointer records to the acquired address, since this would conceal the details of the address management mechanism from the application program, thus precluding a specific application from accessing data improperly (see text of claim 2, page 3).

26. Regarding claim 20, the Applicants teach as **Admitted Prior Art** a computer readable medium storing a program for managing a plurality of data which are used in order to execute an application program for providing services related to communication by a switching system substantially as claimed, the program comprising the steps of:

Art Unit: 2177

- a) a data field storing the plurality of data (see data field in Figure 21, labeled as Prior Art; see also Applicants' specification, page 3, last paragraph through page 4, first paragraph);
- b) an address acquirer acquiring an address of the data in said data field for which an access is requested by the application program (see pointer acquisition step in Figure 21, labeled as Prior Art; see also Applicants' specification, page 3, last paragraph through page 4, first paragraph); and
- c) a lender lending the address of the data to the application program (see pointer lending step in Figure 21, labeled as Prior Art; see also Applicants' specification, page 3, last paragraph through page 4, first paragraph).

The **Admitted Prior Art** fails to explicitly teach a computer readable medium comprising a lending pointer table storing at least one of pointer records having the acquired address and a pointer corresponding to the acquired address, and a lender reading out the pointer from the lending pointer table.

Koyama, however, teaches a computer readable medium comprising a lending pointer table storing at least one of pointer records having the acquired address and a pointer corresponding to the acquired address, and a lender reading out the pointer from the lending pointer table (see Constitution section, page 2; see also text of claim 1, beginning on page 2; see also paragraph [0016], beginning on page 10).

It would have been obvious to one of ordinary skill in the art at the time of the invention to incorporate a lending pointer table for storing pointer records to the acquired address, since this

would conceal the details of the address management mechanism from the application program, thus precluding a specific application from accessing data improperly (see text of claim 2, page 3).

27. Regarding claims 2, 17 and 21, **Koyama** additionally teaches a data management apparatus, method and computer readable medium further comprising a reader receiving the lent pointer from the application program, reading out the address corresponding to the lent pointer from the lending pointer table, reading out the data storing the read address in said data field, and giving the read data to the application program (see Constitution section, page 2; see also text of claim 1, beginning on page 2; see also paragraph [0016], beginning on page 10).

28. Claim 3 is rejected under 35 U.S.C. 103(a) as being unpatentable over **Admitted Prior Art** in view of **Koyama** (Japanese Patent Publication 7-295814) as applied to claims 1, 2, 16, 17, 20 and 21 above, and further in view of **Cabrera et al.** (U.S. Patent 6,029,160).

29. Regarding claim 3, **Admitted Prior Art** and **Koyama** teach a data management apparatus substantially as claimed.

Neither **Admitted Prior Art** nor **Koyama** explicitly teaches a data management apparatus including a deleter deleting a data from said data field, and a record deleter deleting the pointer record having the address of the data which is deleted by said deleter from said lending pointer table.

Cabrera et al., however, teaches a data management apparatus including a deleter deleting a data from said data field, and a record deleter deleting the pointer record having the address of the

Art Unit: 2177

data which is deleted by said deleter from said lending pointer table (see disclosure of analogous functionality at col. 9, lines 26-38 and col. 10, lines 28-35).

It would have been obvious to one of ordinary skill in the art at the time of the invention to delete data and references to said data, since this is the only way for obsolete data to be removed from the system.

30. Claim 4 is rejected under 35 U.S.C. 103(a) as being unpatentable over **Admitted Prior Art** in view of **Koyama** (Japanese Patent Publication 7-295814) in view of **Cabrera et al.** (U.S. Patent 6,029,160) as applied to claim 3 above, and further in view of **Hacherl et al.** (U.S. Patent 5,787,442).

31. Regarding claim 4, **Admitted Prior Art**, **Koyama** and **Cabrera et al.** teach a data management apparatus substantially as claimed.

None of **Admitted Prior Art**, **Koyama** nor **Cabrera et al.** explicitly teach a data management apparatus including an invalidity informer for informing the invalidation of the pointer in the pointer record which is deleted by said record deleter.

Hacherl et al., however, teaches a data management apparatus including an invalidity informer for informing the invalidation of the pointer in the pointer record which is deleted by said record deleter (see Abstract; see also col. 1, line 34 through col. 2, line 13).

It would have been obvious to one of ordinary skill in the art at the time of the invention to inform interested processes when a pointer record is deleted, since this allows the interested process to take appropriate actions to endure that referential integrity is maintained (see col. 1, line 34 through col. 2, line 13).

32. Claims 5 and 6 are rejected under 35 U.S.C. 103(a) as being unpatentable over **Admitted Prior Art** in view of **Koyama** (Japanese Patent Publication 7-295814) as applied to claims 1, 2, 16, 17, 20 and 21 above, and further in view of **Hale et al.** (U.S. Patent 5,502,836).

33. Regarding claims 5 and 6, **Admitted Prior Art** and **Koyama** teach a data management apparatus substantially as claimed.

Neither **Admitted Prior Art** nor **Koyama** explicitly teaches a data management apparatus including a relocater for relocating data stored in said data field.

Hale et al., however, teaches a data management apparatus including a relocater for relocating data stored in said data field (see col. 2, lines 55-57), and an address updater detecting the address of the data which is relocated and updating the detected address to an address after the relocation process (see col. 2, lines 57-59), and wherein the address updater waits until any pending read requests of the data being relocated are completed (see disclosure that the relocation process waits for a lull in I/O before executing, col. 12, lines 10-64).

It would have been obvious to one of ordinary skill in the art at the time of the invention to provide the claimed relocation functionality, since this allows data to be reorganized on the disks such that it can be accessed more efficiently.

34. Claim 7 is rejected under 35 U.S.C. 103(a) as being unpatentable over **Admitted Prior Art** in view of **Koyama** (Japanese Patent Publication 7-295814) as applied to claims 1, 2, 16, 17, 20 and 21 above, and further in view of **Watson et al.** (U.S. Patent 4,755,939).

35. Regarding claim 7, **Admitted Prior Art** and **Koyama** teach a data management apparatus substantially as claimed.

Neither **Admitted Prior Art** nor **Koyama** explicitly teaches a data management apparatus including a record deleter that performs garbage collection.

Watson et al., however, teaches a data management apparatus including a record deleter that performs garbage collection (see disclosure that the system detects when no more pointers to a given cell exist, and then performs garbage collection, freeing the cell (see Abstract, et seq.).

It would have been obvious to one of ordinary skill in the art at the time of the invention to institute garbage collection, since it is important to provide some means of reclaiming cells which are no longer required, so that they can be re-allocated for further use (see col. 1, lines 21-23).

36. Claims 8, 18 and 22 are rejected under 35 U.S.C. 103(a) as being unpatentable over **Admitted Prior Art** in view of **Koyama** (Japanese Patent Publication 7-295814) as applied to claims 1, 2, 16, 17, 20 and 21 above, and further in view of **Hitz et al.** (U.S. Patent 5,819,292).

37. Regarding claims 8, 18 and 22, **Admitted Prior Art** and **Koyama** teach a computer readable medium, data management apparatus and method substantially as claimed.

Neither **Admitted Prior Art** nor **Koyama** explicitly teaches a computer readable medium, data management apparatus and method further comprising a data setting area management table, an allocation controller and an adder.

Hitz et al., however, teaches a computer readable medium, data management apparatus and method comprising a data setting area management table storing information related to the use-condition of each data setting area (see disclosure of the block map and inode file, both of which provide analogous functionality, col. 9, line 50 through col. 10, line 48), an allocation controller referring to said data setting area management table, and determining at least one of empty data setting areas in order to allocate a data requested to be added (see disclosure that new data is only written to unallocated blocks, col. 4, lines 14-16), and an adder storing the data requested to be added to at least one of the empty data setting areas which is determined by said allocation controller (see disclosure that new data is only written to unallocated blocks, col. 4, lines 14-16).

It would have been obvious to one of ordinary skill in the art at the time of the invention to implement a system of file space allocation mapping such as that claimed, since the operating system

must have a way of accessing data for reading, and also must have to capability to write new data to memory/disk space that is unoccupied.

38. Claim 11 is rejected under 35 U.S.C. 103(a) as being unpatentable over **Admitted Prior Art** in view of **Koyama** (Japanese Patent Publication 7-295814) in view of **Hitz et al.** (U.S. Patent 5,819,292) as applied to claims 8, 18 and 22 above, and further in view of **Hacherl et al.** (U.S. Patent 5,787,442).

39. Regarding claim 11, **Admitted Prior Art**, **Koyama** and **Hitz et al.** teach a data management apparatus substantially as claimed.

None of **Admitted Prior Art**, **Koyama** nor **Hitz et al.** explicitly teach a data management apparatus wherein the lender informs the application program that there is no data for which access is requested by the application program when the detected use-condition is under the condition of deleted.

Hacherl et al., however, teaches a data management apparatus wherein the calling application is notified when requested data has been deleted (see Abstract; see also col. 1, line 34 through col. 2, line 13).

It would have been obvious to one of ordinary skill in the art at the time of the invention to inform interested processes when a pointer record is deleted, since this allows the interested process

to take appropriate actions to endure that referential integrity is maintained (see col. 1, line 34 through col. 2, line 13).

Conclusion

40. The prior art made of record and not relied upon is considered pertinent to applicant's disclosure.

Korty (U.S. Patent 5,021,946) teaches a method for selecting the sizes and ordering of the extents used to construct a file, a segment, or a virtual space of a computer system file.

Eilert et al. (U.S. Patent 5,095,420) teaches a technique for mapping a data space to an address space.

Johnson et al. (U.S. Patent 5,175,852) teaches a distributed file management system with a plurality of nodes and a plurality of files.

McCrory (U.S. Patent 5,751,979) teaches a video controller that enables applications operating in a protected, multiprocessing system to update a video memory at native speeds.

Provino et al. (U.S. Patent 5,799,314) teaches a method of controlling the mapping of data buffers for heterogeneous programs.

Mori et al. (U.S. Patent 5,806,058) teaches a database management system for accessing data stored in a database via an index.

Tatsumi et al. (U.S. Patent 5,832,491) teaches a system for relocating records in a database having a prime region and an overflow region in parallel with a service processing.

Gladney (U.S. Patent 6,044,378) teaches a system for determining a relationship between a first and second data element by using a relationship element.

Saboff (U.S. Patent 6,199,203) teaches a system for managing the memory of a software component such that the state of the software component is preserved after an update to the software component.

Oguchi et al. (U.S. Patent 6,304,912) teaches a communication apparatus containing a first table having entries each storing a data-link-layer path to a second communication apparatus.

Forin (U.S. Patent 6,360,220) teaches a lock-free method of accessing information in an indexed computer data structure which includes a lookup procedure, an insertion procedure, a removal and replacement procedure and a release procedure.

Capps (U.S. Patent 6,397,311) teaches a system and method of defragmenting a file system.

Bohannon et al. (U.S. Patent 6,449,623) teaches a method of detecting and recovering from data corruption in a database.

Smith (U.S. Patent 6,516,329) teaches a method of handling a search through the use of a page index.

Yamada (Japanese Patent Publication JP410275082) teaches a system that dynamically generates an object on a computer across a network by making use of a remote procedure.

Kodera (Japanese Patent Publication JP02001005704) teaches a system for increasing the efficiency of data access by generating a data table of unique keys permitting direct access to data in a database that an application requires.

Jodeit ("Storage Organization in Programming Systems") teaches a system of program and data representation for use in a computer system.

Ben-Amram et al. ("On Pointers Versus Addresses") investigates the cost of random access to memory.

Buhr et al. (« μ Database : Parallelism in a Memory-Mapped Environment ») investigates the behavior of data structures and their algorithms, both parallel and sequential, in a memory-mapped environment.

The following reference, while not qualifying as prior art, is also of interest:

Choy (U.S. Patent 6,581,060) teaches a system for protecting records in a relational database management system in accordance with non-RDBMS access control rules.


Any inquiry concerning this communication or earlier communications from the examiner should be directed to Luke S. Wassum whose telephone number is 703-305-5706. The examiner can normally be reached on Monday-Friday 8:30-5:30, alternate Fridays off.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, John E. Breene can be reached on 703-305-9790. The fax phone number for the organization where this application or proceeding is assigned is 703-872-9306.

In addition, INFORMAL or DRAFT communications may be faxed directly to the examiner at 703-746-5658.

Customer Service for Tech Center 2100 can be reached during regular business hours at (703) 306-5631, or fax (703) 746-7240.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see <http://pair-direct.uspto.gov>. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free).


Luke S. Wassum
Art Unit 2177

Notice of References Cited	Application/Control No. 09/668,995	Applicant(s)/Patent Under Reexamination KOJIMA ET AL.	
	Examiner Luke S. Wassum	Art Unit 2177	Page 1 of 2

U.S. PATENT DOCUMENTS

*		Document Number Country Code-Number-Kind Code	Date MM-YYYY	Name	Classification
	A	US-4,755,939 A	07-1988	Watson, Paul	707/206
	B	US-5,021,946 A	06-1991	Korty, Joseph A.	707/205
	C	US-5,095,420 A	03-1992	Eilert et al.	711/209
	D	US-5,175,852 A	12-1992	Johnson et al.	707/8
	E	US-5,502,836 A	03-1996	Hale et al.	711/170
	F	US-5,751,979 A	05-1998	McCrary, Duane J.	345/803
	G	US-5,787,442 A	07-1998	Hacherl et al.	707/201
	H	US-5,799,314 A	08-1998	Provino et al.	709/230
	I	US-5,806,058 A	09-1998	Mori et al.	707/2
	J	US-5,819,292 A	10-1998	Hitz et al.	707/203
	K	US-5,832,491 A	11-1998	Tatsumi et al.	707/101
	L	US-6,029,160 A	02-2000	Cabrera et al.	707/1
	M	US-6,044,378 A	03-2000	Gladney, Henry Martin	707/103R

FOREIGN PATENT DOCUMENTS

*		Document Number Country Code-Number-Kind Code	Date MM-YYYY	Country	Name	Classification
	N	JP 07295814 A	11-1995	JP	Koyama	G06F 9/40
	O	JP 10275082 A	10-1998	JP	Yamada	G06F 9/44
	P	JP 2001005704 A	01-2001	JP	Kodera	G06F 12/00
	Q					
	R					
	S					
	T					

NON-PATENT DOCUMENTS

*		Include as applicable: Author, Title Date, Publisher, Edition or Volume, Pertinent Pages)
	U	Jodeit, J.G. "Storage Organization in Programming Systems", Communications of the ACM, Vol. 11, No. 11, pp. 741-746, November 1968.
	V	Ben-Amram, A.M. and Z. Galil "On Pointers Versus Addresses", Journal of the ACM, Vol. 39, No. 3, pp. 617-648, Juny 1992.
	W	Buhr, P.A., A.K. Goel, N. Nishimura and P. Ragde " Database: Parallelism in a Memory-Mapped Environment", Proceedings o the ACM Symposium on Parallel Algorithms and Architectures, pp. 196-199, June 1996.
	X	

*A copy of this reference is not being furnished with this Office action. (See MPEP § 707.05(a).)
Dates in MM-YYYY format are publication dates. Classifications may be US or foreign.

Notice of References Cited	Application/Control No. 09/668,995	Applicant(s)/Patent Under Reexamination KOJIMA ET AL.	
	Examiner Luke S. Wassum	Art Unit 2177	Page 2 of 2

U.S. PATENT DOCUMENTS

*		Document Number Country Code-Number-Kind Code	Date MM-YYYY	Name	Classification
	A	US-6,199,203 B1	03-2001	Saboff, Michael L.	717/168
	B	US-6,304,912 B1	10-2001	Oguchi et al.	709/238
	C	US-6,360,220 B1	03-2002	Forin, Alessandro	707/8
	D	US-6,397,311 B1	05-2002	Capps, James A.	711/165
	E	US-6,449,623 B1	09-2002	Bohannon et al.	707/202
	F	US-6,516,329 B1	02-2003	Smith, Kim C.	715/501.1
	G	US-6,581,060 B1	06-2003	Choy, David Mun-Hien	707/9
	H	US-			
	I	US-			
	J	US-			
	K	US-			
	L	US-			
	M	US-			

FOREIGN PATENT DOCUMENTS

*		Document Number Country Code-Number-Kind Code	Date MM-YYYY	Country	Name	Classification
	N					
	O					
	P					
	Q					
	R					
	S					
	T					

NON-PATENT DOCUMENTS

*		Include as applicable: Author, Title Date, Publisher, Edition or Volume, Pertinent Pages)
	U	
	V	
	W	
	X	

*A copy of this reference is not being furnished with this Office action. (See MPEP § 707.05(a).)
Dates in MM-YYYY format are publication dates. Classifications may be US or foreign.

Please type a plus sign (+) → +

Patent and Trademark Office U.S. DEPARTMENT OF COMMERCE

1449/PTO

U.S. Department of Commerce
Patent and Trademark Office

Application No. :

Filing Date :

First Named Inventor: Katsue KOJIMA Et al.

Group Art Unit :

Examiner Name :

Attorney Docket No. : FUJY 17.788

INFORMATION DISCLOSURE

STATEMENT BY APPLICANT

Sheet 1 of 2

U.S. PATENT DOCUMENTS

Examiner Initials	Cite No. ¹	U.S. Patent Document	Kind Code if known ²	Name of Patentee or Applicant of Cited Document	Date of Publication of Cited Document MM-DD-YYYY	Pages, Columns Lines Where Relevant Passages or Relevant Figures Appear
JSN		5,561,785		International Business Machines Corporation	10-1-96	711/170

FOREIGN DOCUMENTS

Examiner Initials	Cite No. ¹	Foreign Patent Document Office ³ Number ⁴ Kind Code ⁵ (if known)	Country	Name of Patentee or Applicant of Cited Document	Date of Publication of Cited Document MM-DD-YYYY	Pages, Columns Lines Where Relevant Passages or Relevant Figures Appear
JSN	Abstract only	6-214874	JP	TOSHIBA CORP.	08/05/94	606F 12/02
		3-50651	JP	HITACHI, LTD.	03/05/91	606F 12/10
		5-229599	JP	International Business Machines Corporation	10/24/96	606F 12/02

Examiner Signature

Duke S. Wadsworth

Date Considered

28 January 2004

Examiner: Initial if reference considered, whether or not citation is in conformance with MPEP 609. Draw a line through citation if not in conformance and not considered. Include copy of this form with next communication to applicant.

¹ Unique citation designation number. ² See attached Kinds of U.S. Patent Documents. ³ Enter Office that Issued the document, by the two-letter code (WIPO Standard ST.3). ⁴ For Japanese patent documents, the indication of the year of the reign of the Emperor must precede the serial number of the patent document. ⁵ Kind of document by the appropriate symbols as indicated on the document under WIPO Standard ST.1⁶ if possible. ⁶ Applicant is to place a check mark here if English language Translation is attached.

Burden Hour Statement: This form is estimated to take .2 hours to complete. Time will vary depending upon the needs of the individual case. Any comments on the amount of time you are required to complete this form should be sent to the Chief Information Officer, Patent and Trademark Office, Washington, DC 20231. Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of Information unless it displays a valid OMB control number. DO NOT SEND FEES OR COMPLETED FORMS TO THIS ADDRESS. SEND TO: Assistant Commissioner for Patents, Washington, DC 20231.

U.S. PTO
09/668995

DERWENT-ACC-NO: 1996-024611

DERWENT-WEEK: 199603

COPYRIGHT 1999 DERWENT INFORMATION LTD

TITLE: Data address management method in computer - involves
using same point holding predetermined data in pointer
table, when same data is accessed for second time

PATENT-ASSIGNEE: FUJI ELECTRIC CO LTD[FJIE], FUJIFACON CORP[FUJX]

PRIORITY-DATA: 1994JP-0090402 (April 28, 1994)

PATENT-FAMILY:

PUB-NO	PUB-DATE	LANGUAGE	PAGES	MAIN-IPC
JP 07295814 A	November 10, 1995	N/A	006	G06F 009/40

APPLICATION-DATA:

PUB-NO	APPL-DESCRIPTOR	APPL-NO	APPL-DATE
JP 07295814A	N/A	1994JP-0090402	April 28, 1994

INT-CL (IPC): G06F009/35, G06F009/40

ABSTRACTED-PUB-NO: JP 07295814A

BASIC-ABSTRACT:

The method involves setting up a tag name for each variable data which is repeatedly used. The address of each data is stored in a data management table. When a program is processed, in a program counter, it searches for the variable data in the management table.

Corresponding to the tag name, the pointer adjust stored in pointer table is extracted, when same data is accessed for second time. This address denotes the memory location of that particular data.

USE/ADVANTAGE - In factory automation, process automation. Reduces processing work. Increases performance. Facilitates reading of data by simple logic.

CHOSEN-DRAWING: Dwg.1/6

TITLE-TERMS: DATA ADDRESS MANAGEMENT METHOD COMPUTER POINT HOLD
PREDETERMINED

DATA POINT TABLE DATA ACCESS SECOND TIME

ADDL-INDEXING-TERMS:

FACTORY AUTOMATION PROCESS AUTOMATION

DERWENT-CLASS: T01

*1/11/01 Request
Submitted 13 Feb 2001*

EPI-CODES: T01-F03A;

SECONDARY-ACC-NO:

Non-CPI Secondary Accession Numbers: N1996-020748

PTO 2004-1913

Japan Kokai
Japanese Patent Publication
Publication No.: 7 - 295814

ADDRESS MANAGEMENT METHOD FOR DATA INPUT

(De-ta nyuryoku no adoresu kanri hoho)

Shinichiro Koyama

UNITED STATES PATENT AND TRADEMARK OFFICE

Washington D.C.

March 2004

Translated by Schreiber Translations, Inc.

<u>Country</u>	:	Japan
<u>Document No.</u>	:	07-295814
<u>Document Type</u>	:	Patent Publication
<u>Language</u>	:	Japanese
<u>Inventor</u>	:	Shinichiro Koyama
<u>Applicant</u>	:	Fuji Facom Control Co.
<u>IPC</u>	:	G 06 F 9/40; 9/35
<u>Application Date</u>	:	April 28, 1994
<u>Publication Date</u>	:	November 10, 1995
<u>Foreign Language Title</u>	:	De-ta nyuryoku no adoresu kanri hoho
<u>English Title</u>	:	ADDRESS MANAGEMENT METHOD FOR DATA INPUT

[Title of Invention] ADDRESS MANAGEMENT METHOD FOR DATA INPUT

[Summary]

[Purpose] The required data is extracted from the data region when the original program is called out. The overhead during processing is reduced significantly even though the data points are numerous and the performance of the process is not reduced.

[Constitution] During the data extraction process according to the name tag displayed in the flow of the program, a means is provided to judge if the extraction is the first time in the flow of the said program or it is more than 2 times. Also, a pointer Table is provided to keep the pointers showing the access route of the memory region where the data is stored actually and this data corresponds to the name tag. The pointer that is possessed during the data extraction according to the initial name tag is stored in the pointer Table corresponding to that name tag. When the accessing is more than 2 times, the data is accessed using the pointer held in the pointer Table. /2

[Scope of Patent Claims]

[Claim 1] The address management method of the data input is characterized in that during the data extraction process

¹ the numbers in the margin indicate pagination in foreign text

according to the name tag displayed in the flow of the program,

1st, 2nd, 3rd
1st, 2nd, 3rd (a name tag) is attached to the data used repeatedly in the program, the target data inside the ring that is previously set in the data management Table is extracted from the data storage region or that data is displayed and,

A means is provided to judge if the extraction is the first time in the flow of the said program or it is more than 2 times and,

A pointer Table is provided to keep the pointer showing the access route of the memory region where the data is stored actually and this data corresponds to the name tag.

The pointer that is possessed during the data extraction according to the initial name tag is stored in the pointer Table corresponding to that name tag. When the accessing is more than 2 times, the data is accessed using the pointer held in the pointer Table.

[Claim 2] The address management method of the input data as stated in Claim 1 is characterized in that the address management mechanism is constructed as the function sub-routine. By setting the mechanism and the region to hold the pointer of the data region in this function, the address management mechanism from the called out side of the program is concealed.

[Detailed explanation of the invention]

[0001]

[Industrial field of use] The invention pertains to the address management method of the data input in the computer program used for control management of the process system as the information data inputted from the computer terminal of the processing system are processed.

[0002]

[Prior Art] In a computer program of a FA (Factory automation) and PA (Process Automation), the ^{search} data inputted from a computer terminal connected to a main computer is processed and the data that is obtained is attached a name (name tag). By drawing the data handled with the program, the data can be secondary ^{pointer to handling program table} processed or when it is image displayed, the data that are inputted from these control devices with the program are handled with some variable number inside the computer.

[0003] The extraction of the data according to the name tag is performed with a pointer in the data region that is in the ring of the data management Table that is defined as the name tag and the real data, that data is read. As shown in Figure 4, the call out routine which is the online display program consists of (a) the name and data identification initial setting, (b) the data extraction program and (c) the data display process, the data that is obtained is stored in the data region of the memory. As

shown in figure 5, the data that is extracted is attached with a name tag. In each routine of the aforementioned data process program, the relationship between the Table provided in the memory region of each referenced routine is displayed. For example, a certain data is extracted that is attached with a name tag (voltage 1) and the program is displayed on the display device. The data of the name tag (voltage 1) used in the display program is extracted and the ring in this process is explained next.

[0004] In figure 5, the instantaneous registers P1 and P2 of the data storage region is attached with each name tag and are provided in the memory region of the process data input process routine. These are replaced by the data inputted from the processing terminal device based on the program flow shown in figure 4(C). The process routine corresponding to the data of each name tag is based on this input data, the maximum value, the minimum value, the average value and the corrected value of the input data in a certain period are computed and processed. It is replaced with the value from the data storage region of the corresponding memory.

[0005] When the data shown as "voltage 1" in the display device is determined as the average value of the previous voltage 1, it is stored in each name tag storage region of the display program as the call out original routine. The value of the average

value register of "voltage 1" from the process data input process routine memory region is extracted according to the name tag "voltage 1". This is stored in the data storage region of the called out routine memory region. The display program is displayed as the average value of the "the voltage 1" on the display device using this data.

[0006] To perform the aforementioned data extraction, the name tag management table and the data management table are provided in the memory region of each of the data extraction routine. In this name tag management table, a name tag attached to a special data is defined corresponding to the data number inside the "data management table". Also, "in the data management table", the average value of the voltage 1 corresponds to the "voltage 1" of each name tag. When this is the multiple accurate integer type, this data is defined as the pointer of the address in the actual memory that is stored.

[0007] Figure 6 is a flow chart showing the procedure of the extraction of the data in the mechanism shown in figure 5. The data is extracted based on the ring in the management table indicating the name tag according to this procedure.

[0008]

[The problems resolved by the invention] The handling of the data in the program is simple by using the data management according to the name tag. On the other hand, the ring in the

management table of the data is not directly related to the display process so there is an overhead. In particular, when there is a number of data points, the performance is deteriorated which is a drawback.

[0009] The purpose of the invention is to offer an address management method of the input data where the required data is extracted from the data region when the original program is called out. The overhead during processing is reduced significantly even though the data points are numerous and the performance of the process is not reduced. /3

[0010]

[Means for resolving the problems] To achieve the aforementioned purpose, the address management method of the invention for the data input is characterized in that during the data extraction process according to the name tag displayed in the flow of the program, a name tag is attached to the data used repeatedly in the program, the target data inside the ring that is previously set in the data management Table is extracted from the data storage region or the data is displayed and,

A means is provided to judge if the extraction is the first time in the flow of the said program or it is more than 2 times.

A means is provided to hold the pointer in the region where the required data is actually stored.

[0011] A pointer Table is provided to keep the pointer showing the access route of the memory region where the data is stored actually and this data corresponds to the name tag. The pointer that is possessed during the data extraction according to the initial name tag is stored in the pointer Table corresponding to that name tag. When the accessing is more than 2 times, the data is accessed using the pointer held in the pointer Table.

[0012]

[Action] According to the aforementioned means, when a certain data is extracted from the data storage of the memory according to the program with the original name tag, since the access pointer of the data storage region that is possessed based on the ring in the data management Table is stored in the pointer table corresponding to the name tag, during data accessing, the data corresponding to the name tag can be extracted from the memory directly according to the contents indicated in the pointer table, the data is inputted into the program.

[0013]

[Implementation examples] In an implementation example of the input method for the data extraction of the program based on the invention, the indicated data is searched and extracted with the name tag. The process for storing the data in the data storage region of the called out routine is performed. The state of the pointer and the address that is held in the management table

provided in each routine are shown in figure 1. The method of the invention is explained according to the diagram shown below.

[0014] The basic flow of the online display program which is the called out routine in the program of the implementation example is the same as figure 4(a). The basic flow of the data extraction routine which is the data extraction process is the same as figure 4(b). Then, the basic flow of the process data input process routine where the data inputted from the process terminal is stored in the data region is all the same as the conventional technology which is shown in figure 4(C).

[0015] Thus, in this implementation example as described above, the process data is read according to 3 programs, "the online display program", "the data extraction routine" and "the process data input process routine", these are the called out routine. The process data input process routine inputs the instantaneous value data of the control and view at a certain period into the computer and the maximum, minimum and average value are computed and processed. These are kept as the data in a certain region of the process data input process routine memory region. Also, "the online display program" is the program for displaying the data indicated with a name tag called "voltage 1" on the screen at a certain period. The average value of the voltage 1 in the name tag called "voltage 1" is attached during the design of the system program.

[0016] First, "the online display program" calls out the "data extraction routine", the "voltage 1" is indicated as the name tag and the character row is indicated as the type of data. The data search routine of "the data extraction routine" is the program for performing the flow process of figure 2. When the data classification indication is judged with the process for classifying the initial data type, the present call out is recognized as the initial call out. Then, the name tag management table is searched by the character row called "voltage 1". The pointer LJK of the data management table corresponding to the "voltage 1" and the 10 of the management number of the name tag "voltage 1" are possessed. Then, the data management table is referenced using the possessed pointer LJK. The address X31, X32, X33 are possessed as the access pointer of the region where the data actually exists. In addition, the region of the data input function is kept in the memory. The value of "the voltage 1" corresponding to the management number 10 of the name tag management table exists in the pointer table opened in the region of this input region, addresses X31, X32 and X33 of this memory are set as the pointers. When the setting of this pointer table is completed, the character row "voltage 1" of name tag storage region is replaced in the 10 of the management number, the data type is changed into the pointer type from the character row. Finally, the data is read from the

region indicated with the possessed pointer and written into the "online display program".

[0017] "The online display program" displays the value of the "voltage 1" on the screen and when the value of the "voltage 1" is displayed, in the data search routine of the "data extraction routine", since the information of storage in the name tag storage region is modified to the pointer type according to the aforementioned, the data is searched from the character row, the data from the indicated region of the direct pointer is read out and outputted in the "online display program".

[0018] Next, the data extraction process is made into a sub-program. Another implementation example of the method of the invention is shown in figure 3, the process is called out as the function in the process of the main call out program process. The invention is explained below. The called out program of figure 3 called out the data input function where the name tag of the data is attached as a number, the extracted input of the data during the flow process is performed.

[0019] The data input function is the sub-program for performing the data search process displayed in figure 2 of the data extraction routine pertaining to the aforementioned 1st invention. The data of the name tag attached as the transfer number is extracted, the process is executed to transfer this. "The data input function" is a classified data which is the

pointer type so the data input from now is more than 2 times. The pointer of the data region is recognized as in possession. Therefore, the 10 in the management number of the possession is stored in the key, this pointer is referenced, this pointer is from the data region stored in number 10 of the pointer table. The process returns to the called out program where the value corresponding to the "voltage 1" is read out. /4

[0020] The action of more than 3 times is the same as the action of 2 times. This is repeated until the "online display program" is completed. The "data input function" according to the action as mentioned above performs the extraction of the data indicated according to the name tag.

[0021] According to the 1st invention, after the initial search of the search information stored in the name tag storage region, the process is modified to the pointer type to access the appropriate data directly. A route is not required to search the data from the character row. Since the data from the region indicated with a direct pointer is read out, a lot of data can be repeated and the overhead can be reduced according to the ring in the management table and the high speed data read out is possible, an effective method can be offered.

[0022] Also, according to the 2nd invention, in order to extract the data via the "data input function", since the process inside the "data input function" from the original call out program is

concealed, the read out of the present data is the first time and the read out of 2 or more times is not required. The data read out programming is possible according to a simple logic.

[Brief explanation of the diagrams]

[Figure 1] The diagram is used to explain the ring of the data extraction in the implementation example of the invention.

[Figure 2] This is the diagram showing the flow of the program searching the data from the tag name according to the invention.

[Figure 3] This is a flow diagram of the program searching for the data from the tag name according to the data input function.

[Figure 4] This is the constitution explanation diagram of the data management program according to the tag name.

[Figure 5] This is the constitution diagram of the data management method according to the conventional tag name.

[Figure 6] This is the diagram for explaining the inner part processing procedure of the conventional method.

【図2】

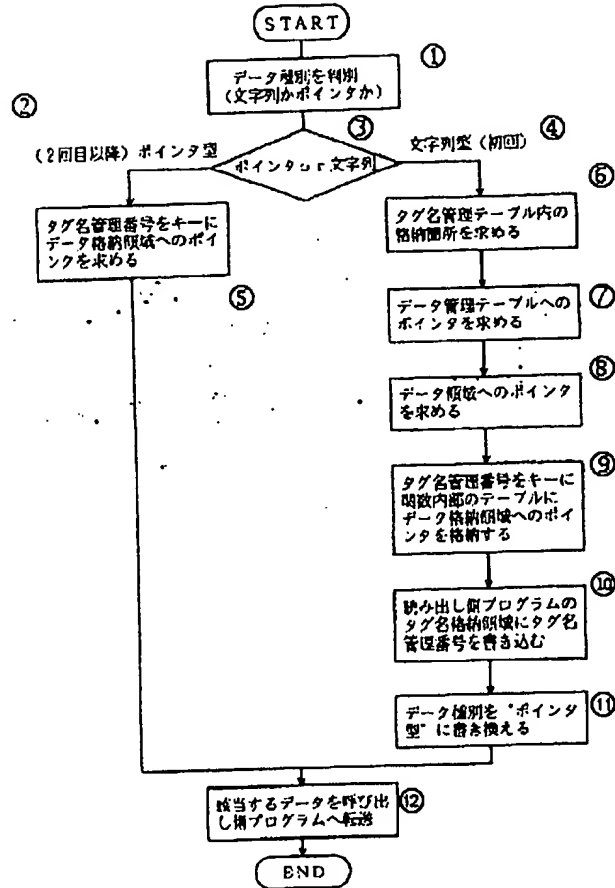


Figure 2

START

Keys:

- 1 - data classification is judged (whether the character row is a pointer)
- 2 - pointer type (2 or more times)
- 3 - pointer or character row
- 4 - character row type (initial time)
- 5 - pointer of the data storage region is obtained and the tag name management number is keyed.

6 - the storage location inside the tag name management table is obtained

7 - the pointer of the data management table is obtained

8 - the pointer to the data region is obtained

9 - the tag name management number is keyed, the pointer to the data storage area is stored in the function Table

10 - the tag name management number is written into the tag name storage region of the read out side of the program

11 - the data classification is converted to the "pointer type"

12 - the said data is called out and transferred to the said program

END

【図3】

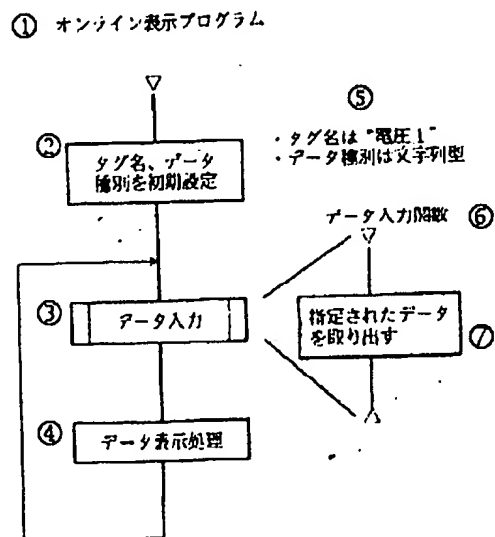


Figure 3

Keys:

- 1 - on line display program
- 2 - tag name, data classification is set initially
- 3 - data input
- 4 - data display process
- 5 - tag name is "voltage 1"; data classification is the character type
- 6 - data input function
- 7 - indicated data is extracted

【図6】

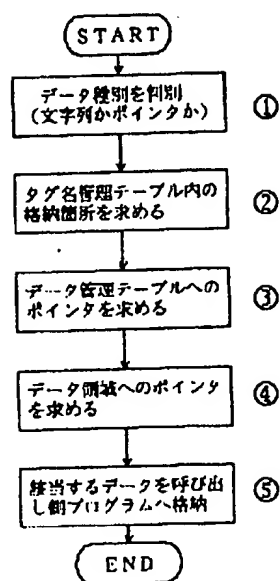


Figure 6

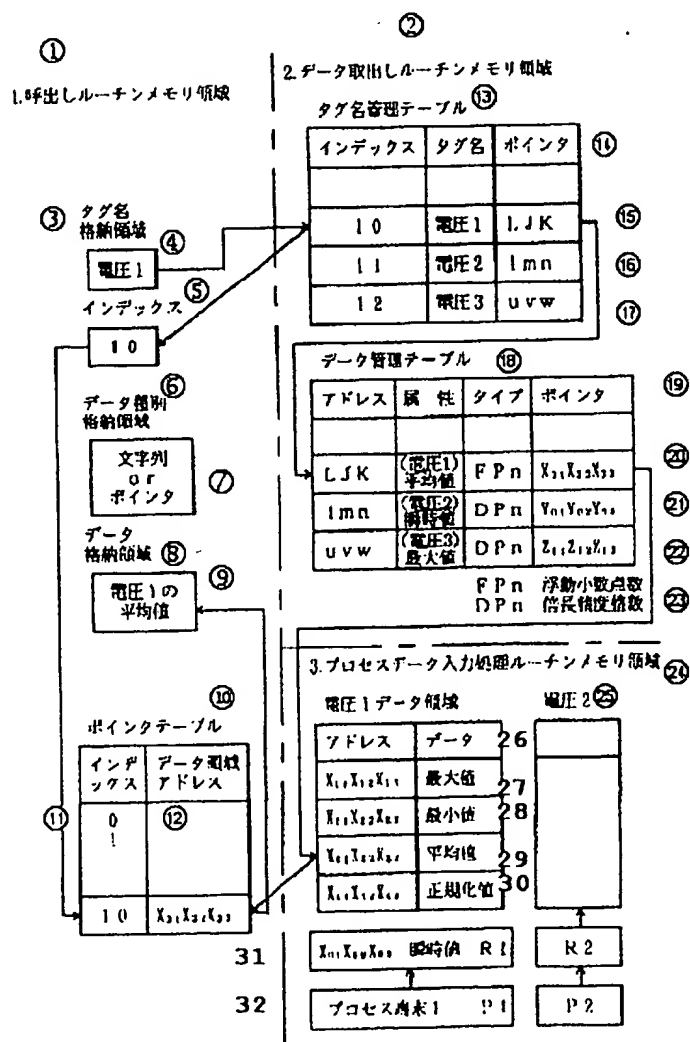
START

- 1 - data classification is judged (whether the character row is the pointer)

- 2 - the storage location inside the tag name management table is obtained
 - 3 - The pointer of the data management table is obtained
 - 4 - The pointer of the data region is obtained
 - 5 - the appropriate data is called out and stored in the program
- END

/5

【図1】



Keys:

- 1 - Called out routine memory region
- 2 - data called out routine memory region
- 3 - tag name storage region
- 4 - voltage 1
- 5 - index
- 6 - data identification storage region
- 7 - character row or pointer
- 8 - data storage region
- 9 - average value of voltage 1
- 10 - pointer table
- 11 - Index, 12 - Data region address
- 13 - name tag management Table
- 14 - Index, Name tag, Pointer
- 15 - Voltage 1, 16 - Voltage 2, 17 - Voltage 3
- 18 - Data management Table
- 19 - Address; Relation; Type; Pointer
- 20 - (Voltage 1) average value
- 21 - (Voltage 2) instantaneous value
- 22 - (Voltage 3) maximum value
- 23 - Fpn, floating decimal points; DPn multiple accurate integer
- 24 - 3. Process data input process routine memory region
- 25 - voltage 1 data region, voltage 2
- 26 - address; data

- 27 - maximum value
- 28 - minimum value
- 29 - average value
- 30 - corrected value
- 31 - instantaneous value
- 32 - process terminal 1

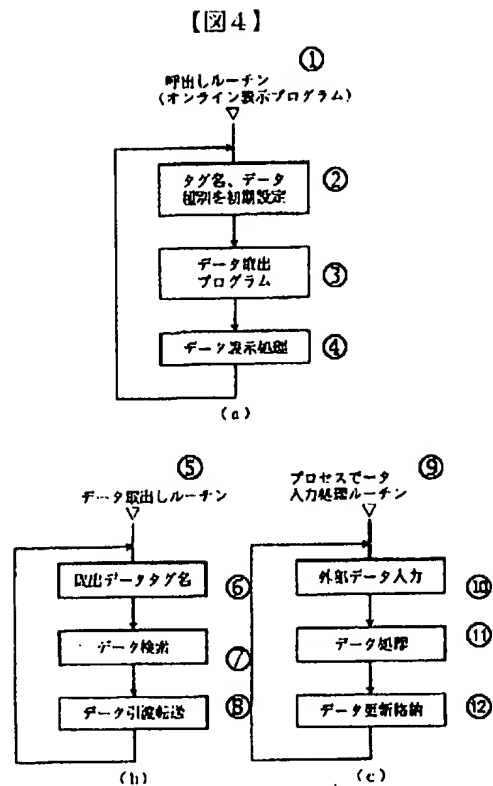


Figure 4

(a)

- 1 - called out routine (online display program)
- 2 - tag name, data classification are set initially
- 3 - data extraction program
- 4 - data display process

(b)

- 5 - Data extraction routine
- 6 - extraction data tag name
- 7 - data search
- 8 - data transfer
- 9 - data input process routine in the process
- 10 - external part data input
- 11 - data process
- 12 - data renewal storage

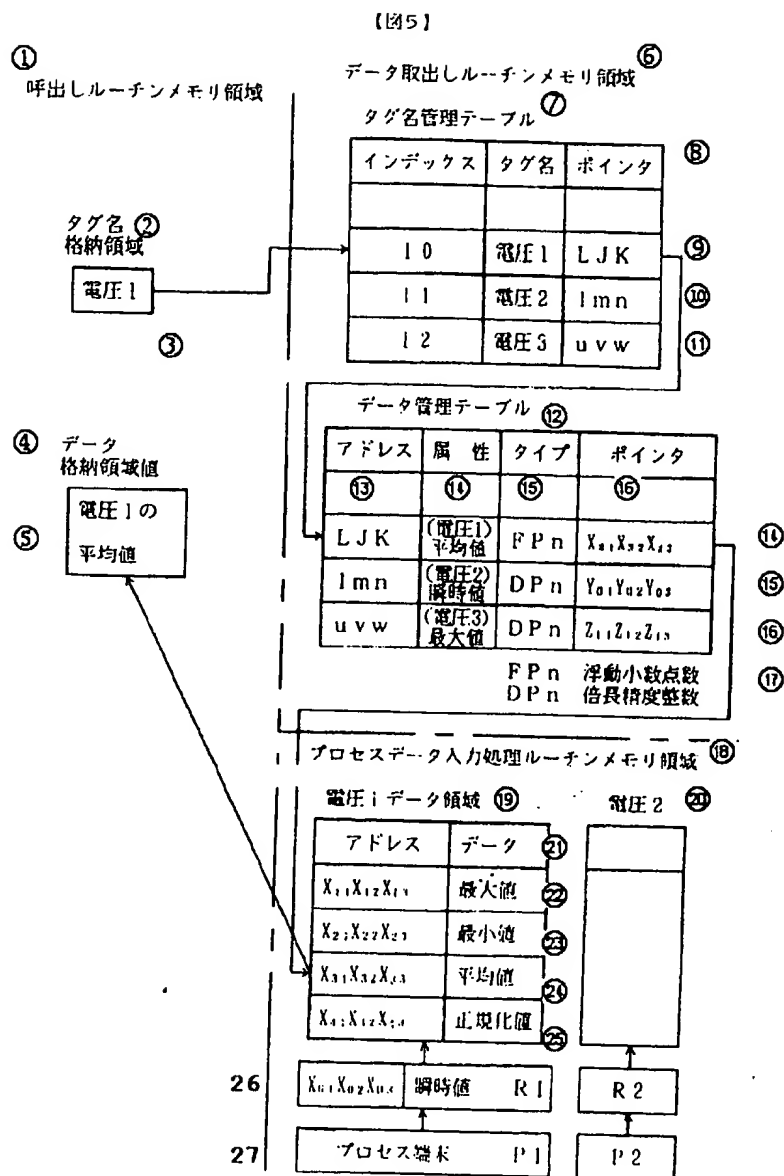


Figure 5

- 1 - Called out routine memory region
- 2 - tag name storage region
- 3 - voltage 1
- 4 - data storage region value

5 - average value of voltage 1
 6 - data called out routine memory region
 7 - tag management Table
 8 - index (10,11,12); tag name; pointer (LJK..uvw)
 9 - Voltage 1, 10 - Voltage 2, 11 - Voltage 3
 12 - Data management Table
 13 - Address; 14 - Relation; 15 - Type; 16 - Pointer
 LJK - (Voltage 1) average value
 lmn - (Voltage 2) instantaneous value
 uvw - (Voltage 3) maximum value
 17 - Fpn, floating decimal number; DPn multiple accurate integer
 18 - Process data input process routine memory region
 19 - voltage 1 data region, 20 - voltage 2
 21 - address; data
 22 - maximum value
 23 - minimum value
 24 - average value
 25 - corrected value
 26 - instantaneous value R1
 27 - process terminal P1

【特許請求の範囲】

【請求項1】プログラム中で繰り返して使用するデータにタグ名を付して変数と同等に扱い、データ管理テーブルにあらかじめ設定されたリンクをたどって目的のデータをデータ格納域から取り出して加工あるいは表示の処理を行うデータ処理装置プログラムにおけるデータ入力

のアドレス管理方法であって、プログラムのフローに表われる前記タグ名によるデータの取り出し処理が、該プログラムのフローにおいて初回

であるか、2回目以降かを識別する手段と、前記タグ名に対応するデータが、実際に格納されている記憶領域へのアクセスルートを示すポインタを保持するポインタテーブルと、を備え、

初回のタグ名によるデータ取り出しの際に獲得した前記ポインタをタグ名に対応させて前記ポインタテーブルに納め、2回目以降のアクセスの際は、前記ポインタテーブルに保持したポインタを用いてデータをアクセスすることを特徴とするデータ入力

のアドレス管理方法。【請求項2】アドレス管理機構を関数サブプログラムとして構成し、この関数内にデータ領域へのポインタを保持する機構および領域を設けることにより、呼び出し側のプログラムからこのアドレス管理機構を隠蔽することを特徴とする請求項1に記載のデータ入力

のアドレス管理方法。

【発明の詳細な説明】

【0001】

【産業上の利用分野】この発明はプロセス・システムの端末装置機器から時々刻々入力される情報データを処理してプロセス・システムの制御管理を行う計算機プログラムにおけるデータ入力

のアドレス管理方法に関する。

【0002】

【従来の技術】FA（ファクトリー・オートメーション）、PA（プロセス・オートメーション）にかかわる計算機プログラムでは、計算機に接続された端末装置機器が入力するデータを処理して得られるデータに名称（タグ名）を付け、プログラムで取り扱うデータを抽象化することにより、データを二次加工するとき、あるいは画面表示する際、プログラム上でこれらの制御機器から入力するデータが、あたかも計算機内部の変数であるかのごとく取り扱う方法がとられる。

【0003】タグ名によるデータの取出しは、タグ名と実際のデータとを定義したデータ管理テーブルのリンクをたどってデータ領域へのポインタを求めてデータを読み込む方法が従来用いられている。図4に、データをタグ名によって処理するデータ処理プログラムを構成するオンライン表示プログラムなどのデータの呼出しルーチン（a）と、データ取出しルーチン（b）、及び端末装置機器から入力されたデータを処理して得たデータをメモリのデータ領域に格納するプロセスデータ入力処理ルーチン（c）の基本的流れを示す。そして図5に、タグ

名を付したデータの取り出しにあたって、上記のデータ処理プログラムの各ルーチンにおいて、参照される各ルーチンのメモリ域に設けられるテーブル間の関連を示し、たとえば、タグ名「電圧1」を付した所定のデータを取り出して表示装置上に表示するプログラムを例に、メモリのデータ格納領域から、表示プログラムで使用するタグ名「電圧1」のデータが抽出されるとき

の処理のリンクを説明する。【0004】図5において、プロセスデータ入力処理ルーチンのメモリ域に設けられた各タグ名を付されたデータ格納域の瞬時値レジスタP1、P2には、図4（c）のプログラムフローに基づきプロセス端末装置機器から入力されるデータによって書き替え更新され、この入力データをもとに、該タグ名のデータに対応する処理ルーチンが所定の期間における入力データの最大値、最小値、平均値、正規化値などを演算処理して求め、メモリの対応するデータ格納域の値と置換している。

【0005】表示装置に「電圧1」として表示するデータを、あらかじめ電圧1の平均値と定めた場合に、呼び出し元ルーチンとしての表示プログラムのタグ名格納領域に格納されているタグ名「電圧1」によって、プロセスデータ入力処理ルーチンメモリ領域から「電圧1」の平均値レジスタの値が取りだされて、呼び出しルーチンメモリ領域のデータ格納領域に収容され、このデータを用いて表示プログラムが表示装置上に「電圧1」の平均値として表示する。

【0006】上記のデータ取り出しを行うため、データ取り出し各ルーチンのメモリ領域には、タグ名管理テーブルとデータ管理テーブルとが設けられている。このタグ名管理テーブルでは、特定のデータに付けたタグ名が「データ管理テーブル」内のどのデータ項目に対応するかを定義付けている。また、「データ管理テーブル」では、電圧1の平均値がタグ名の「電圧1」に相当し、これが倍精度整数型であること、さらにこのデータが実際に格納されているメモリのアドレスがポインタとして定義されている。

【0007】図6は、図5に示す機構におけるデータの取り出しの手順を示すフローチャートである。この手順により、タグ名を指定して管理テーブルのリンクをたどってデータを取り出すことができる。

【0008】

【発明が解決しようとする課題】タグ名によるデータ管理を用いることでプログラム上のデータの取扱は簡単になる反面、データの管理テーブルのリンクをたどる表示処理に直接関係しない処理であるオーバーヘッドがあり、特にデータ点数が多い場合は性能が落ちるという欠点がある。

【0009】本発明は、呼び出し元プログラムがデータ領域から所要のデータを取り出すとき、処理にかかるオーバーヘッドを極力少なくしてデータ点数が多い場合に

も処理の性能が低下しないようにする入力データのアドレス管理方法の提供を目的とする。

【0010】

【課題を解決するための手段】上記の目的達成のため本発明においては、タグ名によって指定されたデータを取り出すプログラムにつきの処理工程を設ける。

・現在のデータの読み出しが初回か、2回目以降かを識別する手段。

・必要なデータの実際に格納されている領域へのポインタを保持する手段。

【0011】要求されたデータに対して管理テーブルに設定されたリンクをたどってデータの取り出しを行う際、初回のデータ入力で獲得したデータ領域へのポインタを保持するためのテーブルを作成し、2回目以降のアクセスからは作成したテーブル内のポインタが指示するメモリ内のデータ領域の値を読み込むようにする。

【0012】

【作用】上記の手段によれば、プログラムが最初にタグ名によって当該のデータをメモリのデータ格納領域から取り出すときに、データ管理テーブルのリンクをたどって獲得したデータ格納領域へのアクセスポインタが、タグ名に対応してポインタテーブルに納められるので、次のデータアクセスに際しては、ポインタテーブルに指示されている内容によって直接タグ名に対応のデータをメモリから取り出してプログラムに入力することができる。

【0013】

【実施例】本発明にもとづくプログラムへのデータ取り出し入力方法の一実施例において、タグ名で指定されたデータを検索して取り出し、呼び出しルーチンのデータ格納領域に格納する処理を遂行する各ルーチンに設けられる管理テーブルに保持されるポインタとアドレスの状態を図1に示し、以下この図によって本発明の方法を説明する。

【0014】本実施例のプログラムにおける呼び出しルーチンとしてのオンライン表示プログラムの基本フローは図4の(a)と同等であり、データを取り出す処理過程であるデータ取り出しルーチンの基本フローも図4の(b)と同等である。そして、プロセス端末から入力されるデータを処理してデータ領域に格納するプロセスデータ入力処理ルーチンの基本フローも図4の(c)に示される従来技術におけると全く同等である。

【0015】上記のように、この実施例では、呼び出しルーチンとしての「オンライン表示プログラム」、「データ取り出しルーチン」および「プロセスデータ入力処理ルーチン」の3つプログラムによってプロセスデータを読み取ってこれを表示するシステムが構成されている。プロセスデータ入力処理ルーチンは、定周期で制御・監視にかかわる瞬時値データを計算機内に取り込んで最大値、最小値、平均値などに演算処理し、プロセスデ

ータ入力処理ルーチンメモリ領域の所定の領域にそれぞれのデータとして保持している。また、「オンライン表示プログラム」は「電圧1」という名前で指定されたデータを定周期で画面に表示するプログラムであり、「電圧1」というタグ名には電圧1の平均値がシステムプログラムの制作にあたって割り付けられているものとする。

【0016】まず、「オンライン表示プログラム」はタグ名として「電圧1」を、データ種別としては文字列型を指定して「データ取り出しルーチン」を呼び出す。

「データ取り出しルーチン」のデータ検索ルーチンは、図2のフローの処理を行うプログラムであり、最初のデータ種別を判別する工程でデータ種別指定が文字列であると判定したときには、今回の呼び出しが初回の呼び出しであることを認識する。そして、「電圧1」という文字列によってタグ名管理テーブルを検索し、タグ名「電圧1」の管理番号の10と「電圧1」に対応するデータのデータ管理テーブルへのポインタLJKとを獲得する。つぎに、獲得したポインタLJKを用いてデータ管理テーブルを参照し、実際にデータが存在する領域へのアクセスポインタとしてのアドレスX31X32X33を獲得する。さらに、メモリにデータ入力関数の領域を確保し、この入力関数の領域に開いたポインタテーブルに、タグ名管理テーブルの管理番号10に対応させて「電圧1」の値が存在するメモリのアドレスX31X32X33をポインタとして設定する。この、ポインタテーブルの設定が終了したとき、タグ名格納領域の文字列「電圧1」を管理番号の10に書換え、データ種別を文字列型からポインタ型に変更する。最後に、獲得したポインタの指す領域からデータを読み出し「オンライン表示プログラム」に引き渡す。

【0017】「オンライン表示プログラム」は、得られた「電圧1」の値を画面に表示するが、次に「電圧1」の値を表示するときには、「データ取り出しルーチン」のデータ検索ルーチンにおいて、タグ名格納領域に格納の情報が上記のようにポインタ型に変更されているので、文字列からデータを検索するルートをとることなく、直接ポインタの指す領域からデータを読み出し「オンライン表示プログラム」に引き渡す。

【0018】次に、データ取り出し工程をサブプログラム化し、メインの呼び出しプログラムの処理の内では関数として呼び出されるようにした本発明の方法の他の実施例を図3に示し、以下にこの発明を説明する。図3の呼び出し元プログラムは、処理フローの中でデータの取り出し入力を必要とする工程において入力対象のデータのタグ名を引数として付してデータ入力関数を呼び出す。

【0019】データ入力関数は、前記第1の発明におけるデータ取り出しルーチンの図2のフローに示されるデータ検索プロセスの処理を行うサブプログラムであり、

10

20

30

40

50

5

引数として付されたタグ名のデータを取出してこれを転送する処理を実行する。「データ入力関数」はデータの種別がポインタ型であることから今回のデータ入力に2回目以降であって、データ領域へのポインタは既に獲得済であることを認識する。したがって、すでに獲得済の管理番号の10をキーとしてポインタテーブルの10番目に格納されているデータ領域へのポインタを参照して「電圧1」に対応する値を読み出して呼び出しプログラムに復帰する。

【0020】3回目以降の動作は2回目の動作と同じであり、これは「オンライン表示プログラム」が終了するまで繰り返される。以上のような動作によって「データ入力関数」はタグ名によって指定されたデータの取り出しを行う。

【0021】

【発明の効果】第一の発明によれば、タグ名格納領域に格納の検索情報がタグ名による最初の検索の後に、直接当該データにアクセス可能とするポインタ型に変更されているので、文字列からデータを検索するルートをとることなく、直接ポインタの指す領域からデータを読み出しているため、多数のデータを繰り返して使用する場合、管理テーブルのリンクをたどるオーバーヘッドを少

6

なくすることで高速なデータ読み出しが可能となるといふ効果が得られる。

【0022】また、第二の発明によれば、データの取り出しは「データ入力関数」を介して行っているため、呼び出し元のプログラムからは「データ入力関数」内の処理が隠蔽されるので、呼び出し元のプログラムは今回のデータの読み出しが初回であるか、2回目以降であるかを意識する必要がなく、簡単なロジックによるデータ読み出しのプログラミングが可能になるといふ効果が得られる。

【図面の簡単な説明】

【図1】この発明の実施例におけるデータ取り出しのリンクを説明する図

【図2】この発明によってタグ名からデータを検索するプログラムのフローを示す図

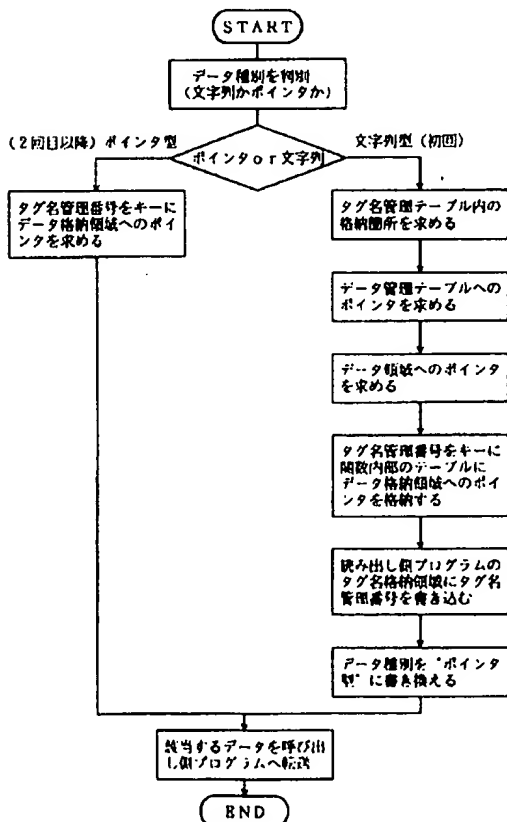
【図3】データ入力関数によってタグ名からデータを検索するプログラムのフロー図

【図4】タグ名によるデータ管理プログラムの構成説明図

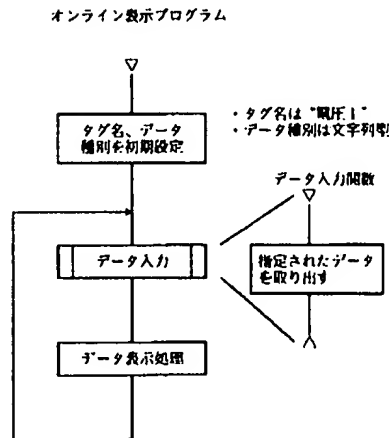
【図5】従来のタグ名によるデータ管理方法の構造図

【図6】従来の方法の内部処理手順を説明する図

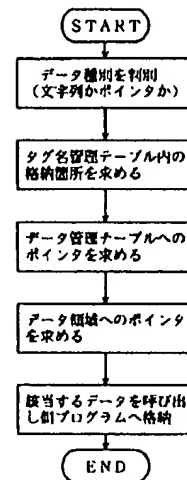
【図2】



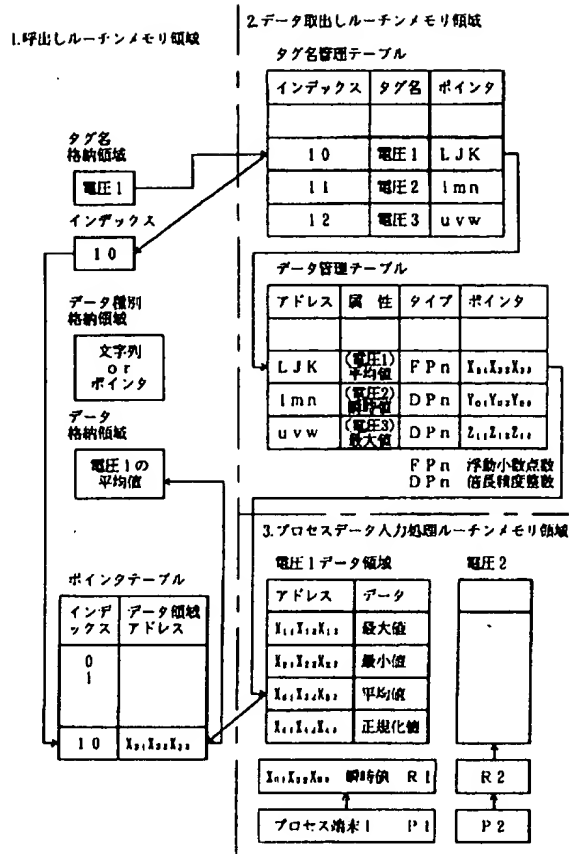
【図3】



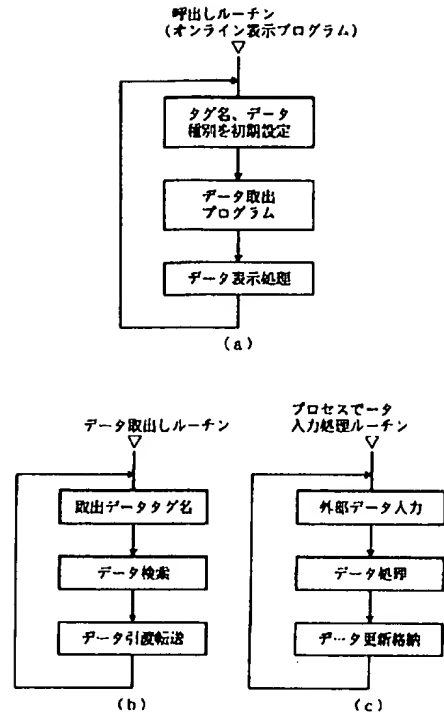
【図6】



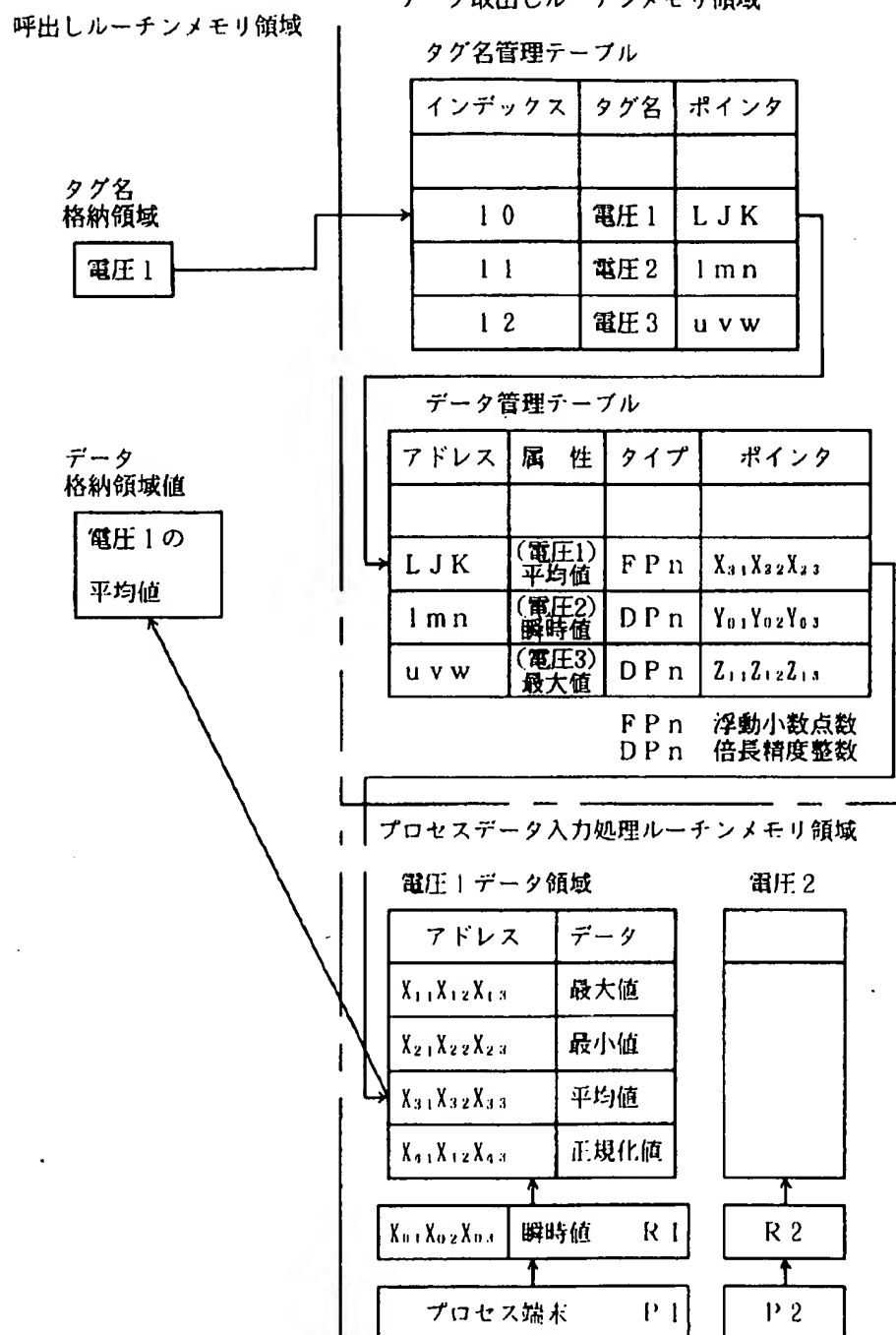
【図1】



【図4】



データ取出しルーチンメモリ領域



PAT-NO: JP410275082A

DOCUMENT-IDENTIFIER: JP 10275082 A

TITLE: SYSTEM AND METHOD FOR REMOTE OBJECT ACCESS

PUBN-DATE: October 13, 1998

INVENTOR-INFORMATION:

NAME

YAMADA, TAKASHI

ASSIGNEE-INFORMATION:

NAME

N T T DATA TSUSHIN KK

COUNTRY

N/A

APPL-NO: JP10020613

APPL-DATE: February 2, 1998

INT-CL (IPC): G06F009/44, G06F009/46, G06F013/00

ABSTRACT:

PROBLEM TO BE SOLVED: To dynamically generate an object on a computer across a network by making use of a remote procedure(RPC) and to access it.

SOLUTION: A client computer 11 has a main program 3 and a conversion logic client stub 8. A server computer 12 has a conversion logic server library 9. In response to a generation request from the main program 3, the server library 9 generates an object 2, assign a unique object ID thereto, and stores this object ID and a pointer into a correspondence table 10 and also returns the object ID to the main program 3. Then, the main program 3 specifies the object ID and issues a process request and a deletion request to the object 2. The server library 9 acquires the pointer corresponding to the object ID from the correspondence table 10 and processes the object 2 that the pointer 2 indicates.

COPYRIGHT: (C)1998,JPO

*XLation Request
submitted 12 Feb 2004*

(19) 日本国特許庁 (J P)

(12) 公開特許公報 (A)

(11) 特許出願公開番号

特開平10-275082

(43) 公開日 平成10年(1998)10月13日

(51) Int.Cl. ⁶	識別記号	F I
G 0 6 F 9/44	5 3 0	G 0 6 F 9/44 5 3 0 M
9/46	3 6 0	9/46 3 6 0 B
13/00	3 5 7	13/00 3 5 7 Z

審査請求 未請求 請求項の数10 O L (全 14 頁)

(21) 出願番号 特願平10-20613

(22) 出願日 平成10年(1998) 2月2日

(31) 優先権主張番号 特願平9-20747

(32) 優先日 平9(1997) 2月3日

(33) 優先権主張国 日本 (J P)

(71) 出願人 000102728

エヌ・ティ・ティ・データ通信株式会社

東京都江東区豊洲三丁目3番3号

(72) 発明者 山田 隆司

東京都江東区豊洲三丁目3番3号 エヌ・

ティ・ティ・データ通信株式会社内

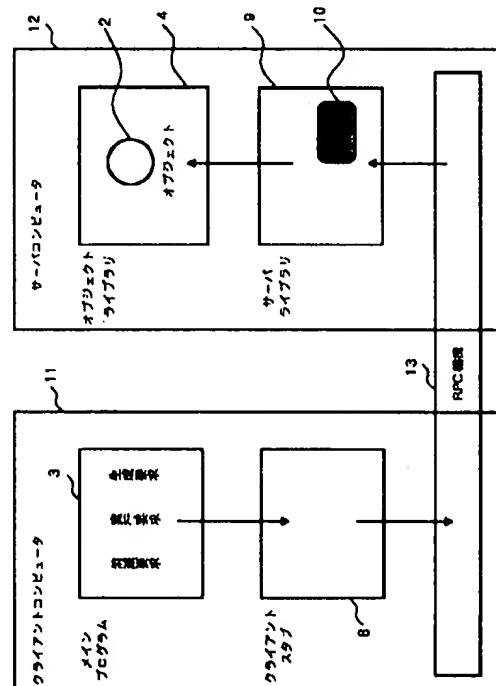
(74) 代理人 弁理士 上村 輝之

(54) 【発明の名称】 リモートオブジェクトアクセスシステム及び方法

(57) 【要約】 (修正有)

【課題】 リモートプロシージャコール (RPC) を利用してネットワークを隔てたコンピュータ上にオブジェクトを動的に生成し且つそれにアクセスする。

【解決手段】 クライアントコンピュータ11はメインプログラム3と変換ロジッククライアントスタブ8とを有する。サーバコンピュータ12は変換ロジックサーバライブラリ9を有する。メインプログラム3からの生成要求に応じて、サーバライブラリ9がオブジェクト2を生成し、これにユニークなオブジェクトIDを割り当て、このオブジェクトIDとポインタとを対応テーブル10に格納し、かつ、そのオブジェクトIDをメインプログラム3に返却する。以後、メインプログラム3は、そのオブジェクトIDを指定してオブジェクト2への処理要求や消滅要求を発行する。サーバライブラリ9は、対応テーブル10からオブジェクトIDに対応するポインタを取得し、そのポインタが指すオブジェクト2に対し処理を行う。



【特許請求の範囲】

【請求項1】 リモートプロシージャコール（RPC）を通じて、クライアントコンピュータがサーバコンピュータ上にオブジェクトを生成し且つそのオブジェクトにアクセスするためのリモートオブジェクトアクセスシステムにおいて、

前記クライアントコンピュータは、

前記クライアントコンピュータ上で生じたオブジェクトの生成及びアクセスのための個々の要求にตอบสนองして、オブジェクトの生成及びアクセスのための個々の関数の呼び出しを前記リモートプロシージャコールを通じて前記サーバコンピュータへ発行するRPCクライアントスタブを備え、

前記サーバコンピュータは、

前記オブジェクトの生成及びアクセスのための関数を有して、前記RPCクライアントスタブからの個々の関数の呼び出しにตอบสนองして、前記サーバコンピュータ上にサーバオブジェクトを生成し且つそのサーバオブジェクトに対するアクセスを実行するRPCサーバライブラリを備え、

RPCクライアントスタブと前記RPCサーバライブラリとは、各サーバオブジェクトに割当てられたユニークなインスタンスIDを前記リモートプロシージャコールを通じてやりとりし、

前記RPCサーバライブラリは、各サーバオブジェクトのインスタンスIDをポインタに変換する手段を有することを特徴とするリモートオブジェクトアクセスシステム。

【請求項2】 リモートプロシージャコール（RPC）を通じて、クライアントコンピュータがサーバコンピュータ上にオブジェクトを生成し且つそのオブジェクトにアクセスするための方法において、

前記クライアントコンピュータ上で生じたオブジェクトの生成及びアクセスのための個々の要求にตอบสนองして、オブジェクトの生成及びアクセスのための個々の関数の呼び出しを前記リモートプロシージャコールを通じて前記サーバコンピュータへ発行するRPCクライアントスタブとして、前記クライアントコンピュータが機能する過程と、

前記オブジェクトの生成及びアクセスのための関数を有して、前記RPCクライアントスタブからの個々の関数の呼び出しにตอบสนองして、前記サーバコンピュータ上にサーバオブジェクトを生成し且つそのサーバオブジェクトに対するアクセスを実行するRPCサーバライブラリとして、前記サーバコンピュータが機能する過程と、

前記RPCクライアントスタブと前記RPCサーバライブラリとは、各サーバオブジェクトに割当てられたユニークなインスタンスIDを前記リモートプロシージャコールを通じてやりとりする過程と、

前記RPCサーバライブラリが、各サーバオブジェクト

のインスタンスIDをポインタに変換する過程と、を有することを特徴とする方法。

【請求項3】 リモートプロシージャコール（RPC）を通じて、クライアントコンピュータがサーバコンピュータ上にオブジェクトを生成し且つそのオブジェクトにアクセスするためのリモートオブジェクトアクセスシステムにおいて、

前記クライアントコンピュータは、

前記クライアントコンピュータ上で生じたオブジェクトの生成及びアクセスのための個々の要求にตอบสนองして、オブジェクトの生成及びアクセスのための個々の関数の呼び出しを前記リモートプロシージャコールを通じて前記サーバコンピュータへ発行するRPCクライアントスタブを備え、

前記サーバコンピュータは、

前記オブジェクトの生成及びアクセスのための関数を有して、前記RPCクライアントスタブからの個々の関数の呼び出しにตอบสนองして、前記サーバコンピュータ上にサーバオブジェクトを生成し且つそのサーバオブジェクトに対するアクセスを実行するRPCサーバライブラリを備え、

前記RPCサーバライブラリは、生成したサーバオブジェクトに対しユニークなインスタンスIDを割り当て、各サーバオブジェクトのインスタンスIDとポインタとの対応テーブルを保有し、且つ、前記生成したサーバオブジェクトに割り当てたインスタンスIDを前記RPCクライアントスタブへ返却し、

前記RPCクライアントスタブは、各サーバオブジェクトのアクセスのための関数呼び出しを発行する時、返却された各サーバオブジェクトのインスタンスIDを前記関数呼び出しに含ませる、ことを特徴とするリモートオブジェクトアクセスシステム。

【請求項4】 請求項3記載のシステムにおいて、

前記クライアントコンピュータが、前記オブジェクト生成のための要求にตอบสนองして生成されるクライアントオブジェクトを更に備え、

前記クライアントオブジェクトは、

(1)その生成時に、前記オブジェクト生成のための関数呼び出しを発行するよう前記RPCクライアントスタブに指示し、且つ、この生成関数の実行の結果返却されたインスタンスIDを前記RPCクライアントスタブより受けて保有し、さらに、

(2)前記オブジェクトのアクセスのための要求にตอบสนองして、前記保有したインスタンスIDを使用して前記アクセスのための関数を発行するよう前記RPCクライアントスタブに指示することを特徴とするリモートオブジェクトアクセスシステム。

【請求項5】 請求項4記載のシステムにおいて、前記クライアントオブジェクトは、オブジェクト種類を示すオブジェクト種別IDを更に有し、このクライアントオ

プロジェクトの生成時に、前記プロジェクト種別IDを使用して前記プロジェクト生成のための関数呼び出しを発行するよう前記RPCクライアントスタブに指示し、前記RPCクライアントスタブは、前記プロジェクト生成のための関数呼び出しを発行する時、前記プロジェクト種別IDを前記関数呼び出しに含ませ、前記RPCサーバライブラリは、オブジェクト種類とオブジェクト種別IDとの対応テーブルを更に有し、前記オブジェクト生成のための関数呼び出しに回答して、前記オブジェクト種別IDに対応したオブジェクト種類に属するオリジナルオブジェクトがサポートするメソッドを有したサーバオブジェクトを生成することを特徴とするリモートオブジェクトアクセスシステム。

【請求項6】 請求項5記載のシステムにおいて、前記クライアントオブジェクトは、前記オリジナルオブジェクトから派生させて構築したものであって、前記オリジナルオブジェクトがサポートするメソッドに対する要求を受け付けることができ、

前記サーバオブジェクトは、前記オリジナルオブジェクトと所定の基本サーバオブジェクトとから派生させて構築したものであって、前記基本サーバオブジェクトがサポートするメソッドに対する要求を受け付けることができ、

前記クライアントオブジェクトは更に、前記オリジナルオブジェクトがサポートするメソッドに対する要求に回答して、そのメソッドに予め割当てられた処理種別を指定して、前記基本オブジェクトがサポートするメソッドに対する処理要求関数の呼出しを発行するよう前記RPCクライアントスタブに指示し、

更に、前記RPCサーバライブラリは、前記基本オブジェクトがサポートするメソッドに対する処理要求関数の呼出しに回答して、前記処理種別を指定して前記基本サーバオブジェクトがサポートするメソッドに対する要求を、前記サーバオブジェクトに発行し、

前記サーバオブジェクトは更に、前記基本サーバオブジェクトがサポートするメソッドに対する要求に回答して、オリジナルオブジェクトがサポートするメソッドの内の前記処理種別に対応するメソッドを実行することを特徴とするリモートオブジェクトアクセスシステム。

【請求項7】 リモートプロシージャコール(RPC)を通じて、クライアントコンピュータがサーバコンピュータ上にオブジェクトを生成し且つそのオブジェクトにアクセスするためのリモートオブジェクトアクセスシステムであって、

前記クライアントコンピュータは、

前記クライアントコンピュータ上で生じたオブジェクトの生成及びアクセスのための個々の要求に回答して、オブジェクトの生成及びアクセスのための個々の関数の呼び出しを前記リモートプロシージャコールを通じて前記サーバコンピュータへ発行するRPCクライアントスタ

ブを備え、

前記サーバコンピュータは、

前記オブジェクトの生成及びアクセスのための関数を有して、前記RPCクライアントスタブからの個々の関数の呼び出しに回答して、前記サーバコンピュータ上にサーバオブジェクトを生成し且つそのサーバオブジェクトに対するアクセスを実行するRPCサーバライブラリを備え、

RPCクライアントスタブと前記RPCサーバライブラリとは、各サーバオブジェクトに割当てられたユニークなインスタンスIDを前記リモートプロシージャコールを通じてやりとりし、

前記RPCサーバライブラリは、各サーバオブジェクトのインスタンスIDをポインタに変換する手段を有するリモートオブジェクトアクセスシステムにおける、前記クライアントコンピュータとしてコンピュータを機能させるためのプログラムを担持したコンピュータ読み取り可能な記録媒体。

【請求項8】 リモートプロシージャコール(RPC)を通じて、クライアントコンピュータがサーバコンピュータ上にオブジェクトを生成し且つそのオブジェクトにアクセスするためのリモートオブジェクトアクセスシステムであって、

前記クライアントコンピュータは、

前記クライアントコンピュータ上で生じたオブジェクトの生成及びアクセスのための個々の要求に回答して、オブジェクトの生成及びアクセスのための個々の関数の呼び出しを前記リモートプロシージャコールを通じて前記サーバコンピュータへ発行するRPCクライアントスタブを備え、

前記サーバコンピュータは、

前記オブジェクトの生成及びアクセスのための関数を有して、前記RPCクライアントスタブからの個々の関数の呼び出しに回答して、前記サーバコンピュータ上にサーバオブジェクトを生成し且つそのサーバオブジェクトに対するアクセスを実行するRPCサーバライブラリを備え、

RPCクライアントスタブと前記RPCサーバライブラリとは、各サーバオブジェクトに割当てられたユニークなインスタンスIDを前記リモートプロシージャコールを通じてやりとりし、

前記RPCサーバライブラリは、各サーバオブジェクトのインスタンスIDをポインタに変換する手段を有するリモートオブジェクトアクセスシステムにおける、前記サーバコンピュータとしてコンピュータを機能させるためのプログラムを担持したコンピュータ読み取り可能な記録媒体。

【請求項9】 リモートプロシージャコール(RPC)を通じて、クライアントコンピュータがサーバコンピュータ上にオブジェクトを生成し且つそのオブジェクトに

5

アクセスするためのリモートオブジェクトアクセスシステムであって、

前記クライアントコンピュータは、

前記クライアントコンピュータ上で生じたオブジェクトの生成及びアクセスのための個々の要求にตอบสนองして、オブジェクトの生成及びアクセスのための個々の関数の呼び出しを前記リモートプロシージャコールを通じて前記サーバコンピュータへ発行するRPCクライアントスタブを備え、

前記サーバコンピュータは、

前記オブジェクトの生成及びアクセスのための関数を有して、前記RPCクライアントスタブからの個々の関数の呼び出しにตอบสนองして、前記サーバコンピュータ上にサーバオブジェクトを生成し且つそのサーバオブジェクトに対するアクセスを実行するRPCサーバライブラリを備え、

前記RPCサーバライブラリは、生成したサーバオブジェクトに対しユニークなインスタンスIDを割り当て、各サーバオブジェクトのインスタンスIDとポインタとの対応テーブルを保有し、且つ、前記生成したサーバオブジェクトに割り当てたインスタンスIDを前記RPCクライアントスタブへ返却し、

前記RPCクライアントスタブは、各サーバオブジェクトのアクセスのための関数呼び出しを発行する時、返却された各サーバオブジェクトのインスタンスIDを前記関数呼び出しに含ませる、リモートオブジェクトアクセスシステムにおける、前記クライアントコンピュータとしてコンピュータを機能させるためのプログラムを担持したコンピュータ読み取り可能な記録媒体。

【請求項10】 リモートプロシージャコール(RPC)を通じて、クライアントコンピュータがサーバコンピュータ上にオブジェクトを生成し且つそのオブジェクトにアクセスするためのリモートオブジェクトアクセスシステムであって、

前記クライアントコンピュータは、

前記クライアントコンピュータ上で生じたオブジェクトの生成及びアクセスのための個々の要求にตอบสนองして、オブジェクトの生成及びアクセスのための個々の関数の呼び出しを前記リモートプロシージャコールを通じて前記サーバコンピュータへ発行するRPCクライアントスタブを備え、

前記サーバコンピュータは、

前記オブジェクトの生成及びアクセスのための関数を有して、前記RPCクライアントスタブからの個々の関数の呼び出しにตอบสนองして、前記サーバコンピュータ上にサーバオブジェクトを生成し且つそのサーバオブジェクトに対するアクセスを実行するRPCサーバライブラリを備え、

前記RPCサーバライブラリは、生成したサーバオブジェクトに対しユニークなインスタンスIDを割り当て、

6

各サーバオブジェクトのインスタンスIDとポインタとの対応テーブルを保有し、且つ、前記生成したサーバオブジェクトに割り当てたインスタンスIDを前記RPCクライアントスタブへ返却し、

前記RPCクライアントスタブは、各サーバオブジェクトのアクセスのための関数呼び出しを発行する時、返却された各サーバオブジェクトのインスタンスIDを前記関数呼び出しに含ませる、リモートオブジェクトアクセスシステムにおける、前記サーバコンピュータとしてコンピュータを機能させるためのプログラムを担持したコンピュータ読み取り可能な記録媒体。

【発明の詳細な説明】

【0001】

【発明の属する技術分野】本発明は、リモートプロシージャコール(遠隔手続き呼び出し；以下、「RPC」と略称する)を用いて、ネットワークを隔てた別のコンピュータ上のオブジェクトを操作するためのリモートオブジェクトアクセスシステムに関する。

【0002】なお、RPCはその実装において、ネットワークからの個々の処理要求を、要求ごとにプロセスを起こし独立したメモリ空間で処理するものと、スレッドやキューイングロジックを用いて共通のメモリ空間で処理するものが存在するが、本方式は後者に適用できる。

【0003】

【従来の技術】手続き型プログラミングにおける呼び出しをネットワークを隔てたコンピュータ間で実施するために、RPCが使われている。RPCはネットワークを隔てた別のコンピュータ上の手続を、あたかもローカルコンピュータ上の手続のように呼出すことができる技術である。RPCを用いてあるコンピュータが別のコンピュータの手続を呼出すと、その別のコンピュータでその手続が実行され、そして、手続の実行が完了すると、その結果が後者から前者へと返される。

【0004】

【発明が解決しようとする課題】ところで、オブジェクト指向プログラミングでは、手続型プログラミングにおけるような手続き呼び出しではなく、オブジェクトの生成、処理要求、消滅のようなオブジェクトに対する操作が行われる。このようなオブジェクト指向プログラミングでのオブジェクト操作に関しても、ネットワークを隔てたコンピュータ間で、別のコンピュータ上のオブジェクトをあたかもローカルコンピュータ上のオブジェクトのように扱えることが望ましい。

【0005】しかしながら、上述した従来のRPCをそのまま利用して、ローカルコンピュータ内でのオブジェクト操作と同様な操作をネットワークを隔てた別のコンピュータ上のオブジェクトに対して行おうと試みた場合、以下のような問題が生じる。

【0006】図1は、ローカルコンピュータ1内でオブジェクト2を操作する場合の通常のプログラムモデルを

10

20

30

40

50

示す。このモデルにおいて、メインプログラム3は次のような操作を行なう。

【0007】1. オブジェクト生成要求を発行し、その結果として、オブジェクトライブラリ4に生成されたオブジェクト2のポインタを取得する。

【0008】2. 以後、取得したポインタを使用して、オブジェクト2に対し所望の処理の実行要求を発行する。

【0009】3. オブジェクト2が不要になったら、取得したポインタを使用してオブジェクト2の消滅要求を

発行する。
【0010】このようにローカルコンピュータ内の通常のプログラムモデルでは、生成したオブジェクトのポインタを使用して、オブジェクトに対する操作がなされる。

【0011】このようなオブジェクト操作を、従来のRPCを利用して、他のコンピュータ上のオブジェクトに対して行おうとする場合、操作要求はRPCの提供する単純な手続き型言語インタフェースに変換せざるを得ない。そのため、例えば、オブジェクトの生成、処理要求、消滅をそれぞれ手続き型言語による手続(関数)として定義し、上述したポインタを関数の引数として設定するような実装を行わざるを得ない。しかし、ネットワークを隔てた別のコンピュータ上にオブジェクトを生成した場合、そのオブジェクトのポインタを取得しても、メモリ空間が異なるために、後の別の関数呼び出しにおいて同じオブジェクトを指定するためにそのポインタを利用できる保証は全くない。つまり、1つの関数呼び出し中ならば、オブジェクトを生成してそのオブジェクトに処理を要求し且つ消滅させることは可能であっても、1つの関数呼び出しでオブジェクトを生成し、別の関数呼び出しでそのオブジェクトに処理を要求し、さらに別の関数呼び出しでそのオブジェクトを消滅させるという操作は、従来のRPCを利用したポインタをやりとりする方式では不可能である。これでは、オブジェクトを必要に応じ動的に生成してアクセスするというオブジェクト指向言語の特長を生かすことができない。

【0012】従って、本発明の目的は、RPCを利用してネットワークを隔てたコンピュータ上にオブジェクトを動的に生成し且つそれにアクセスすることができるリモートオブジェクトアクセスシステムを提供することにある。

【0013】

【課題を解決するための手段】本発明の第1の側面にかかるシステムは、RPCを通じてクライアントコンピュータがサーバコンピュータ上にオブジェクトを生成し且つそのオブジェクトにアクセスする(つまり、処理実行を要求する、及び消滅させる)ためのシステムであって、オブジェクト生成に際しては、クライアントコンピュータが手続き型言語インタフェースの生成要求を発し、

それに応答してサーバコンピュータがオブジェクトを生成すると共にそのオブジェクトにユニークなIDを割り付け、そのIDをクライアントコンピュータにレスポンスとして返す。以後、サーバコンピュータはオブジェクトのIDとポインタとの対応関係を管理する。そして、クライアントコンピュータは、手続き型言語インタフェースのオブジェクトアクセス要求をオブジェクトのIDを指定して発行し、これに応答してサーバコンピュータはそのIDに対応するポインタにより指定されたオブジェクトにアクセスする。

【0014】本発明の第2の側面にかかるシステムは、上記構成に加え、クライアントコンピュータが、サーバコンピュータ内のオブジェクト(サーバオブジェクトという)に対応した仮想的なオブジェクト(クライアントオブジェクトという)を有する。クライアントオブジェクトは、対応するサーバオブジェクトと同一名称のメソッドをサポートし、その生成時及び消滅時には手続き型言語インタフェースの生成要求及び消滅要求を発行し、また、サポートするメソッドの要求を受けると、これを手続き型言語インタフェースの対応する要求に変換する。サーバコンピュータは、クライアントオブジェクトからの要求に応答して、対応するサーバオブジェクトを生成したりアクセスしたりする。本発明の第3の側面にかかるシステムは、第2の側面のシステムの構成に加え、生成するオブジェクトの種類(クラス)を指定できる機構を更に備える。

【0015】なお、本発明のシステムを構成する各コンピュータのプログラムは、ディスク型ストレージ、半導体メモリ、通信回線などの種々の媒体を通じてコンピュータにインストール又はロードすることができる。

【0016】

【発明の実施の形態】以下、本発明の実施の形態を添付図面に従って説明する。尚、以下の説明では、オブジェクトの操作(生成、処理実行、消滅)要求を発するコンピュータを「クライアントコンピュータ」、その要求を受けてオブジェクトの操作を実行するコンピュータを「サーバコンピュータ」と呼ぶ。

【0017】本発明の第1の実施形態に係るリモートオブジェクトアクセスシステムについて図2～図4を参照して説明する。

【0018】この実施形態の概略は次の通りである。すなわち、サーバコンピュータは、クライアントコンピュータからの生成要求に応答したオブジェクトの生成時に、そのオブジェクトに対してユニークなIDを割り付け、そのオブジェクトのポインタとIDとの対応表を保持すると共に、そのIDをクライアントコンピュータにレスポンスとして返す。クライアントコンピュータは、サーバコンピュータからのIDを保持し、以降のサーバコンピュータ上のオブジェクトに対する操作は、オブジェクトのIDを指定して関数を呼び出すことによって実

施する。

【0019】図2は、この実施形態に係るリモートオブジェクトアクセスシステムの構築の基礎となるローカルコンピュータ1内の通常のプログラムモデルを示す。

【0020】図2に示すプログラムモデルは、図1に示した従来のプログラムモデルと比較して明らかなように、メインプログラム3とオブジェクトライブラリ4との間に新たに変換ロジックライブラリ5を有する。この変換ロジックライブラリ5は、オブジェクトの生成、処理実行要求及び消滅というオブジェクト操作のための手続（関数）の集まりであり、メインプログラム3からオブジェクト生成関数の呼び出しを受けると、オブジェクトライブラリ4内にオブジェクト2を生成し、その時にオブジェクト2に対して識別のためのユニークなIDを割り付け、そして、オブジェクトライブラリ4から取得したオブジェクト2のポインタと、オブジェクト2に割り付けたIDとを対応テーブル6に格納して保持及び管理する。

【0021】このプログラムモデルでのオブジェクトの生成とアクセスの動作を以下に説明する。

【0022】1. オブジェクトの生成

メインプログラム3は、オブジェクト生成関数呼び出しを変換ロジックライブラリ5に発行する。変換ロジックライブラリ5は、オブジェクトライブラリ4内にオブジェクト2を生成し、かつユニークなIDを決定し、そして、取得したポインタとIDとを対応テーブル6に格納し、IDをメインプログラム3にIDを返却する。メインプログラム3は、変換ロジックライブラリ5より生成したオブジェクト2のIDを取得する。

【0023】2. オブジェクトの処理実行

メインプログラム3は、取得したIDを引数にセットしてオブジェクト2に対する処理実行要求関数呼び出しを変換ロジックライブラリ5に発行する。変換ロジックライブラリ5は、対応テーブル6からIDに対応するポインタを取得し、そのポインタを使用してオブジェクト2に対して処理実行要求を発行する。

【0024】3. オブジェクトの消滅

メインプログラム3は、オブジェクト2が不要になったら、取得したIDを引数にセットしてオブジェクト消滅関数呼び出しを変換ロジックライブラリ5に発行する。変換ロジックライブラリ5は、対応テーブル6からIDに対応するポインタを取得し、そのポインタを使用してオブジェクト2の消滅要求を発行する。

【0025】図3は、図2に示した通常のプログラムモデルの変換ロジックライブラリ5を基に、本実施形態システムのクライアントコンピュータとサーバコンピュータとにそれぞれ搭載されるプログラムユニットを生成する様子を示す図である。

【0026】図2に示す変換ロジックライブラリ5を公知のRPCコンパイラ7に入力すると、変換ロジックの

RPCクライアントスタブ（以下、単にクライアントスタブと呼ぶ）8と、変換ロジックのRPCサーバライブラリ（以下、単にサーバライブラリと呼ぶ）9とが自動的に生成される。ここで、クライアントスタブ8とサーバライブラリ9とは、RPC機構によって結合されることによって、全体として図2に示す変換ロジックライブラリ5と同じ機能を果たすものである。クライアントスタブ8は、変換ロジックライブラリ5と同じインタフェースをメインプログラム3に提供するものであり、このインタフェースをRPCの手続型インタフェースに変換する機能をもつ。サーバライブラリ9は、変換ロジックライブラリ5内の対応テーブル6に相当する対応テーブル10を保持及び管理しており、変換ロジックライブラリ5本来の機能、つまり、クライアントスタブ8からRPC機構を通じて到来するオブジェクト操作要求に依じて、実際にオブジェクトを生成し、オブジェクトに処理実行要求を発し、かつオブジェクトを消滅させる機能をもつ。

【0027】図4は、上記のクライアントスタブ8とサーバライブラリ9を搭載したコンピュータにより構成される本実施形態システムのプログラムモデルを示す。

【0028】図4に示すように、クライアントコンピュータ11とサーバコンピュータ12とが、ネットワークを隔てて存在し、両者はRPC機構13によって通信可能である。クライアントコンピュータ11は、メインプログラム3と、図3に示したクライアントスタブ8とを備える。サーバコンピュータ12は、オブジェクトライブラリ4と、図3に示したサーバライブラリ9を備える。サーバライブラリ9は、前述したようにオブジェクトライブラリ4内の各オブジェクト2のポインタとIDとの対応テーブル10を有する。前述したように、RPC機構13を通じて結合されたクライアントスタブ8とサーバライブラリ9とが、ちょうど図2に示した通常のプログラムモデルの変換ロジックライブラリ5と同様に機能する。RPC機構13を通じてやりとりされるのは、オブジェクトのポインタではなくIDであるから、クライアントコンピュータ11とサーバコンピュータ12とでメモリ空間が異なることは問題にならない。従って、クライアントコンピュータ11はネットワークを隔てたサーバコンピュータ12上のオブジェクト2を操作することができる。

【0029】次に、図4のシステムにおけるオブジェクト操作について説明する。

【0030】1. オブジェクトの生成

クライアントコンピュータ11のメインプログラム3が、クライアントスタブ8に対してオブジェクト生成関数呼び出しを発行すると、クライアントスタブ8は、この関数呼び出しをRPC機構13を通じてサーバライブラリ9に送る。サーバライブラリ9は、この関数呼び出しに応答して、オブジェクトライブラリ4内にオブジェク

11

ト2を生成し、オブジェクトライブラリ4からオブジェクト2のポインタを取得し、オブジェクト2に対しユニークなIDを割り当て、そのポインタとIDとをテーブル10に格納し、更に、そのIDをRPC機構13を通じてクライアントスタブ8へ返却する。このIDはクライアントスタブ8からメインプログラム3に渡され、メインプログラム3はそのIDを保持する。

【0031】2. オブジェクトの処理実行

メインプログラム3は、取得したIDを引数にセットしてオブジェクト2に対する処理実行要求関数の呼び出しをクライアントスタブ8に発行する。クライアントスタブ8は、その関数呼び出しをサーバライブラリ9に送る。サーバライブラリ9は、その呼び出しに応答して対応テーブル10からそのIDに対応するポインタを取得し、そのポインタによって特定されるオブジェクト2に対して処理の実行要求を発行する。

【0032】3. オブジェクトの消滅

メインプログラム3は、オブジェクト2が不要になったら、取得したIDを引数にセットしてオブジェクト消滅要求関数呼び出しをクライアントスタブ8に発行する。クライアントスタブ8は、その関数呼び出しをサーバライブラリ9に送る。サーバライブラリ9は、対応テーブル10からそのIDに対応するポインタを取得し、そのポインタによって特定されるオブジェクト2に対して消滅要求を発行する。

【0033】以下に、本実施例システムにおける変換ロジックライブラリ5の実装例を示す。オブジェクト指向言語の1つであるC++の例えば「ObjectA」というクラスについての実装例を説明する。ここで、ObjectAは、「Action1」「Action2」「Action3」という3つのメソッドを持つものとする。なお、簡単のためObjectAの生成、消滅及びメソッドにはパラメータがないものとする。

【0034】変換ロジックライブラリ5を実装する場合、手続型言語であるC言語を用いて、オブジェクト操作関数として以下の5つの関数を用意する。

【0035】1. オブジェクト生成関数「ObjectACreate(ID, ret)」

これはクラスObjectAのインスタンス(オブジェクト)を生成するための関数である。この関数は、クラスObjectAのインスタンスを生成すると、そのインスタンスのポインタとそれに割り当てたIDとを対応テーブル10に格納し、そのIDと生成結果retを呼び出し元へ返却する。

【0036】2. オブジェクト消滅関数「ObjectADelete(ID, ret)」

これは、クラスObjectAのインスタンス(オブジェクト)を消滅させるための関数である。この関数は、渡されたIDを基に対応テーブル10から対応するインスタンスのポインタを取得し、そのポインタにより特定

12

されるインスタンスを削除し、対応テーブル10からエントリを削除し、その結果retを呼び出し元へ返却する。

【0037】3. Action1要求関数「ObjectAAction1(ID, ret)」

これは、メソッドAction1をインスタンスに対して要求するための関数である。この関数は、渡されたIDを基に対応テーブル10から対応するインスタンスのポインタを取得し、そのポインタにより特定されるインスタンスに対してAction1を発行し、そのAction1の結果を結果retとして呼び出し元へ返却する。

【0038】4. Action2要求関数「ObjectAAction2(ID, ret)」

5. Action3要求関数「ObjectAAction3(ID, ret)」 発行するメソッドがそれぞれAction2、Action3である以外は、ObjectAAction1(ID, ret)と同じである。

【0039】以上のCの関数を図2に示した変換ロジックライブラリ5として実装し、これを図3に示したようにコンパイルしてできたクライアントスタブ8とサーバライブラリ9とを図4に示すようにネットワーク上のコンピュータ11、12に搭載することにより、クライアントコンピュータ11からの要求でサーバコンピュータ12上にクラスObjectAのインスタンスを生成し、それにAction1、Action2又はAction3を要求したり、消滅させたりすることが可能になる。

【0040】上述した第1の実施形態によると、リモートコンピュータ上にオブジェクトを生成し、処理を要求し、消滅させることができるため、以下のような効果がある。

【0041】(1) オブジェクトに対する処理要求の結果としてオブジェクト内部に生成した情報をリモートマシン上に保持できる。

【0042】(2) オブジェクト内部に生成した情報を、独立した処理要求によって取り出すことができる。

【0043】(3) 複数のコンピュータがリモートコンピュータ上のオブジェクトを共有できる。

【0044】次に本発明の第2の実施形態に係るリモートオブジェクトアクセスシステムについて図5、図6を参照して説明する。

【0045】この実施形態の概略は以下の通りである。サーバコンピュータ上のオブジェクト(サーバオブジェクトと呼ぶ)に対応して、それと同じインタフェースをもつオブジェクト(クライアントオブジェクトと呼ぶ)がクライアントコンピュータ上に生成される。ここで、サーバオブジェクトは「真」のオブジェクトであり、一方、クライアントオブジェクトはインタフェースだけがサーバオブジェクトと同じ「仮想的」なオブジェクトで

あるといえる。クライアントコンピュータ内のメインプログラムがクライアントオブジェクトに操作すると、それに対応したサーバオブジェクトの操作がサーバコンピュータ上で実行される。第1の実施形態では、メインプログラムからのオブジェクト操作はC言語のような手続き型言語で行う必要があったが、第2の実施形態では、オブジェクト指向言語で行うことができる。

【0046】図5は、この実施形態に係るリモートオブジェクトアクセスシステムの構築の基礎となるローカルコンピュータ内の通常のプログラムモデルを示す。

【0047】ローカルコンピュータ21は、図2に示したモデルと同様にメインプログラム3、変換ロジックライブラリ5及びオブジェクトライブラリ4を有する他、メインプログラム3と変換ロジックライブラリ5との間にクライアントオブジェクトライブラリ14を有する。このクライアントオブジェクトライブラリ14は、各サーバオブジェクト2に対応したクライアントオブジェクト15を有する。

【0048】以下、上記モデルでのオブジェクト操作を説明する。

【0049】1. オブジェクトの生成

メインプログラム3が、クライアントオブジェクトライブラリ14内にクライアントオブジェクト15を生成する。クライアントオブジェクト15は、生成処理の中で変換ロジックライブラリ5に対してオブジェクト生成関数の呼び出しを発行する。この呼び出しに回答して変換ロジックのライブラリ5は、サーバオブジェクト2を生成し、オブジェクト2のポインタを取得し、オブジェクト2にユニークなIDを割り付け、ポインタとIDを対応テーブル6に格納し、そして、IDをクライアントオブジェクト15に返却する。クライアントオブジェクト15はそのIDを属性として保持する。

【0050】2. オブジェクトの処理実行

メインプログラム3は、クライアントオブジェクト15に処理の実行を要求する。クライアントオブジェクト15は、属性として保持したIDを引数にセットして、サーバオブジェクト2に対する処理実行要求関数の呼び出しを、変換ロジックライブラリ5へ発行する。変換ロジックライブラリ5は、対応テーブル6からIDに対応するポインタを取得し、そのポインタにより特定されるサーバオブジェクト2に対して処理実行要求を発行する。

【0051】3. オブジェクトの消滅

メインプログラム3は、不要になったクライアントオブジェクト15を消滅させる。クライアントオブジェクト15は、消滅処理の中で、属性として保持したIDを引数にセットしてオブジェクト消滅関数の呼び出し変換ロジックライブラリ5へ発行する。変換ロジックライブラリ5は、対応テーブル6からIDに対応するポインタを取得し、そのポインタが指すサーバオブジェクト2に消

滅要求を発行する。

【0052】図6は、図5に示したモデルを基礎にして構築された本実施例システムのアプログラムモデルを示す。

【0053】図示のように、クライアントコンピュータ31は、メインプログラム3と、クライアントオブジェクトライブラリ14と、クライアントスタブ8とを有する。サーバコンピュータ32は、サーバライブラリ9と、オブジェクトライブラリ4とを有する。ここで、クライアントスタブ8とサーバライブラリ9は、図3に示したように、変換ロジックライブラリ5を公知のRPCコンパイラでコンパイルすることにより生成されたものである。

【0054】本実施例システムでのオブジェクト操作は以下の通りである。

【0055】1. オブジェクトの生成

メインプログラム3は、オブジェクトライブラリ14内にクライアントオブジェクト15を生成する。クライアントオブジェクト15は、生成処理の中でクライアントスタブ8に対してオブジェクト生成関数呼び出しを発行する。クライアントスタブ8は、その関数呼び出しをRPC機構13を通じてサーバライブラリ9に送る。サーバライブラリ9は、その呼び出しに回答してサーバオブジェクト2を生成し、オブジェクト2のポインタを取得し、オブジェクト2にユニークなIDを割り付け、そのポインタとIDを対応テーブル10に格納し、そしてIDをクライアントスタブ8に返す。クライアントスタブ8はそのIDをクライアントオブジェクト15に渡し、クライアントオブジェクト15はそのIDを属性として保持する。

【0056】2. オブジェクトの処理実行

メインプログラム3は、クライアントオブジェクト15に処理実行を要求する。クライアントオブジェクト15は、属性として保持したIDを引数にセットして処理実行要求関数呼び出しをクライアントスタブ8を通じてサーバライブラリ9に発行する。サーバライブラリ9は、対応テーブル10からIDに対応するポインタを取得し、そのポインタが指すサーバオブジェクト2に対して処理実行要求を発行する。

【0057】2. オブジェクトの消滅

メインプログラム3は、不要になったクライアントオブジェクト15を消滅させる。クライアントオブジェクト15は、消滅処理の中で、属性として格納したIDを引数にセットしてオブジェクト消滅関数呼び出しを、RPCクライアントスタブ8を通じてサーバライブラリ9に発行する。サーバライブラリ9は、対応テーブル10からIDに対応するポインタを取得し、そのポインタが指すサーバオブジェクト2に消滅要求を発行する。

【0058】以下に、本実施形態システムの実装例を示す。

【0059】第1の実施形態の実装例と同様に、C++の「ObjectA」をサーバオブジェクトのクラスとし、それが「Action1」「Action2」「Action3」のメソッドを持つものとして、その「ObjectA」についての本実施形態の実装例を説明する。

【0060】まず、第1の実施形態の実装例と同様に、C言語の5つの関数「ObjectACreate(ID, ret)」「ObjectADelete(ID, ret)」「ObjectAAction1(ID, ret)」「ObjectAAction2(ID, ret)」「ObjectAAction3(ID, ret)」を、変換ロジックライブラリ5として用意する。

【0061】次に、サーバオブジェクトのクラスObjectAに対応したクライアントオブジェクトのクラスObjectBを用意する。なお、メインプログラム内でObjectAを使用しない場合、ObjectBの名称はObjectAの名称と同一であっても構わない。このObjectBには、ObjectAがサポートするメソッドと同一名称のメソッドを用意し、各メソッドが上記のC関数を呼び出すように実装する。そのメソッドは以下の通りである。

【0062】1. コンストラクタ（生成時処理）「ObjectB::ObjectB」

これは、生成関数ObjectACreate(ID, ret)を呼び出し、返却されたIDを内部に格納する。

【0063】2. デストラクタ（消滅時処理）「ObjectB::~ObjectB」

これは、内部を保持しているIDを使用して、消滅関数ObjectADelete(ID, ret)を発行する。

【0064】3. 「ObjectB::Action1」

内部保持しているIDを使用して、Action1要求関数呼び出しObjectAAction1(ID, ret)を発行する。

【0065】4. 「ObjectB::Action2」

5. 「ObjectB::Action3」

これらは、発行する関数呼び出しがそれぞれObjectAAction2(ID, ret)、ObjectAAction3(ID, ret)である以外は、Action1と同じ。

【0066】以上の第2の実施形態によると、次の利点を得られる。

【0067】(1) ネットワーク上に配置したいオブジェクトから、機械的にクライアントオブジェクトを生成できるため、簡単にネットワークプログラムが生成できる。

【0068】(2) ネットワーク上のオブジェクトをローカルコンピュータ内のオブジェクトと同様に、ポインタで制御できる。

【0069】(3) ローカルコンピュータ内のオブジェクト管理ロジックをリモートコンピュータ上のオブジェクトに対しても適用できる。

【0070】(4) 本実施形態の拡張方式として、クライアントオブジェクトを全くの仮想的なものとせず部分的に実処理を実装することにより、サーバコンピュータとクライアントコンピュータへの負荷分散が実現できる。

【0071】次に本発明の第3の実施形態に係るリモートオブジェクトアクセスシステムについて、図7、図8を参照して説明する。

【0072】この実施形態は、上述の第2の実施形態に、操作対象となるオブジェクトの種別（クラス）を任意に指定できる機構を付加したものである。操作対象のオブジェクトのクラスを変更したい場合、第2の実施形態では変換ロジックライブラリを再構築しなければならないが、本実施形態によればその必要がない。

【0073】図7は本実施形態に係るリモートオブジェクトアクセスシステムのプログラムモデルを示す。

【0074】クライアントコンピュータ41は、メインプログラム3と、オブジェクトライブラリ51と、変換ロジックのRPCクライアントスタブ（以下、単にクライアントスタブ）53とを備える。オブジェクトライブラリ51には、クライアントオブジェクト52と、これに割り付けられたID（以下、インスタンスIDという）と、そのオブジェクト種別（クラス）を示す種別IDとが保持される。

【0075】サーバコンピュータ42は、変換ロジックのRPCサーバライブラリ（以下、単にサーバライブラリという）54と、オブジェクトライブラリ56とを備える。オブジェクトライブラリ56にはサーバオブジェクト57が保持される。サーバオブジェクト57にはクライアントオブジェクト52が1対1で対応しているが、その具体的な構成は後に説明する。サーバライブラリ54には、サーバオブジェクト57のポインタとインスタンスIDとの対応テーブル10、及びサーバオブジェクト57のオブジェクト種別IDと種類（クラス）との対応テーブル55とが保持される。

【0076】クライアントスタブ53とサーバライブラリ54は、以下のような変換ロジックライブラリを公知のRPCコンパイラによりコンパイルして生成したものである。この変換ロジックライブラリは次の3つの関数をもつ。

【0077】(1) オブジェクト生成関数

この関数は、クライアントオブジェクト52からの呼び出しに含まれているオブジェクト種別IDを基にテーブル55を参照してそのオブジェクト種別IDに対応した

種類(クラス)を認識し、その種類のサーバオブジェクト57をオブジェクトライブラリ56に生成し、そのオブジェクト57に対しユニークなインスタンスIDを割り当て、そのオブジェクト57のインスタンスIDとポインタとをテーブル10に格納し、更に、そのインスタンスIDをクライアントオブジェクト52に返却する。なお、ここで生成されるサーバオブジェクト57は、後述するように基本サーバオブジェクトから派生しさせて構築したものである。

【0078】(2) オブジェクト消滅関数

この関数は、クライアントオブジェクト52からの呼び出しに含まれているインスタンスIDを基にテーブル10を参照してそのインスタンスIDに対応するポインタを取得し、そのポインタが指すサーバオブジェクト57を消滅させる。

【0079】(3) 処理要求関数

この関数は、クライアントオブジェクト52からの呼び出しに含まれているインスタンスIDを基にテーブル10を参照してそのインスタンスIDに対応するポインタを取得し、そのポインタが指すサーバオブジェクト27

に対し、後述する基本サーバオブジェクトがサポートするメソッドを要求する。このメソッドの要求には、後述する処理種別の指定が含まれている。

【0080】図8は、クライアントオブジェクト52とサーバオブジェクト57の構成を示す。

【0081】図8において、基本サーバオブジェクト61は、予め用意された1つのクラスのサーバオブジェクトであり、一つのメソッド60をサポートしている。オリジナルオブジェクト65は、メインプログラム3が生成しようと意図しているオブジェクトであり、ここでは例として3つのメソッド62、63、64をサポートしていることにする。

【0082】図示のようにクライアントオブジェクト52は、オリジナルオブジェクト25から派生させて構築したものであり、構築時にオリジナルオブジェクト25に対して割り付けられたオブジェクト種別IDを属性として持つ。このクライアントオブジェクト52は、図中でオリジナルオブジェクト65と同じ形態に描かれているように、オリジナルオブジェクト65とインタフェースをメインプログラム3に対し提供するものである。このクライアントオブジェクト52は仮想的なオリジナルオブジェクト65ということができ、次のような振舞をする。

【0083】(1) メインプログラム3からの生成要求に応答して生成処理を行い、生成処理では、上述した変換ロジックライブラリに対して、属性として持っているオブジェクト種別IDを引数にセットして上記オブジェクト生成関数の呼び出しを発行し、そして、変換ロジックライブラリから返却されたインスタンスIDを属性として保持する。

【0084】(2) メインプログラム3からオリジナルオブジェクト65のサポートするメソッド62、63又は64の要求を受けると、変換ロジックライブラリに対して、属性として持っているインスタンスIDと、要求されたメソッド62、63又は64に割り付けられた処理種別とを引数にセットして上記処理要求関数の呼び出しを発行する。なお、メソッド62、63、64の処理種別は、クライアントオブジェクト52とサーバオブジェクト57との間で構築時に割り付けられたものである。

【0085】(3) メインプログラムからの消滅要求に応答して消滅処理を行い、消滅処理では、変換ロジックライブラリに対して、属性として持っているインスタンスIDを引数にセットして上記オブジェクト消滅関数の呼び出しを発行する。

【0086】図8に示すように、サーバオブジェクト57は、オリジナルオブジェクト65と基本サーバオブジェクト61の2つのオブジェクトから派生させて構築したものであり、外部に対するインタフェースとして基本サーバオブジェクト61がサポートするメソッド60を有し、このメソッド60の外部から遮蔽された内部に、オリジナルオブジェクト65がサポートするメソッド62、63、64を有し、これらのメソッド62、63、64は前述した処理種別に対応付けられている。

【0087】このサーバオブジェクト57は、変換ロジックライブラリから基本サーバオブジェクト60のサポートするメソッド60を要求されると、その要求に含まれる処理種別に対応したメソッド62、63又は64を実行する。

【0088】以上のモデルにより、メインプログラム3は、自コンピュータ内に所望の種類(クラス)のオブジェクト(オリジナルオブジェクト)を生成するのと同じ操作で、ネットワーク上に当該クラスの真のオブジェクト(サーバオブジェクト57)を生成でき、かつ、自コンピュータ内の所望のオブジェクトに所望のメソッドを要求するのと同じ操作で、ネットワーク上のサーバオブジェクト57に所望のメソッドを実行させることができる。

【0089】本実施形態によれば次の利点が得られる。

【0090】(1) ネットワーク上に配置したいオブジェクトから、機械的にサーバオブジェクト及びクライアントオブジェクトを生成できるため、簡単にネットワークプログラムが生成できる。

【0091】(2) ネットワーク上のオブジェクトをコンピュータ内のオブジェクトと同様に、ポインタで制御できる。

【0092】(3) オブジェクトの種類を追加しない限り、オブジェクトのメソッドの変更などがあつたとしても、変換ロジックライブラリを再構築しなくてもよい。

【0093】以上、本発明の幾つかの好適な実施形態を

10

20

30

40

50

説明したが、これに変更、修正、改良などを加えた他の種々の形態でも本発明を実施することができる。例えば、図4、6、7に示した実施形態において、メインプログラム3の中にクライアントスタブ8、53を構築することも可能である。

【図面の簡単な説明】

【図1】従来のローカルコンピュータにおける通常のプログラムモデルを示すブロック図である。

【図2】本発明の第1の実施形態に係るリモートオブジェクトアクセスシステムの構築の基礎となるローカルコンピュータでの通常のプログラムモデルを示すブロック図である。

【図3】図2に示す変換ロジックライブラリからRPCコンパイラにより変換ロジックのRPCクライアントスタブとRPCサーバライブラリを生成する工程を示す流れ図である。

【図4】第1の実施形態に係るリモートオブジェクトアクセスシステムのプログラムモデルを示すブロック図である。

【図5】本発明の第2の実施形態に係るリモートオブジェクトアクセスシステムの構築の基礎となるローカルコンピュータでの通常のプログラムモデルを示すブロック図である。

【図6】第1の実施形態に係るリモートオブジェクトアクセスシステムのプログラムモデルを示すブロック図である。

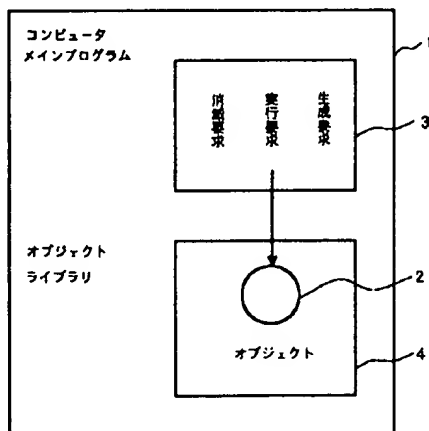
【図7】本発明の第3の実施形態に係るリモートオブジェクトアクセスシステムのプログラムモデルを示すブロック図である。

【図8】第3の実施形態におけるオブジェクトの構成を示す図である。

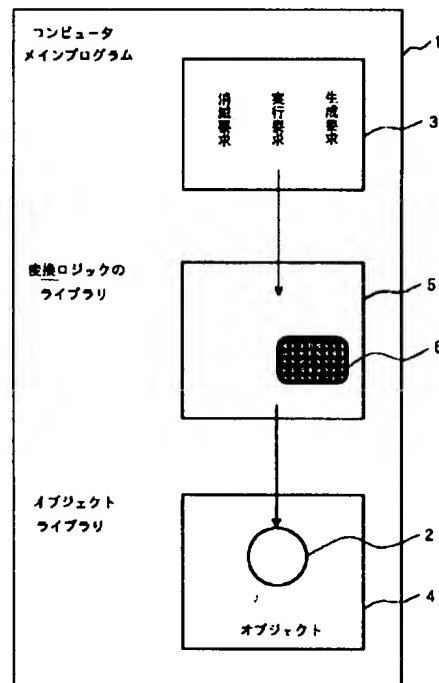
【符号の説明】

- 2、57 サーバオブジェクト
- 3 メインプログラム
- 4、56 オブジェクトライブラリ
- 5 変換ロジックライブラリ
- 6、10 オブジェクトのポインタとIDの対応テーブル
- 8、53 変換ロジックのRPCクライアントスタブ
- 9、54 変換ロジックのRPCサーバライブラリ
- 11、31、41 サーバコンピュータ
- 12、32、42 サーバコンピュータ
- 13 RPC機構
- 14、51 クライアントオブジェクトライブラリ
- 15、52 クライアントオブジェクト
- 55 オブジェクトの種別IDと種類の対応テーブル

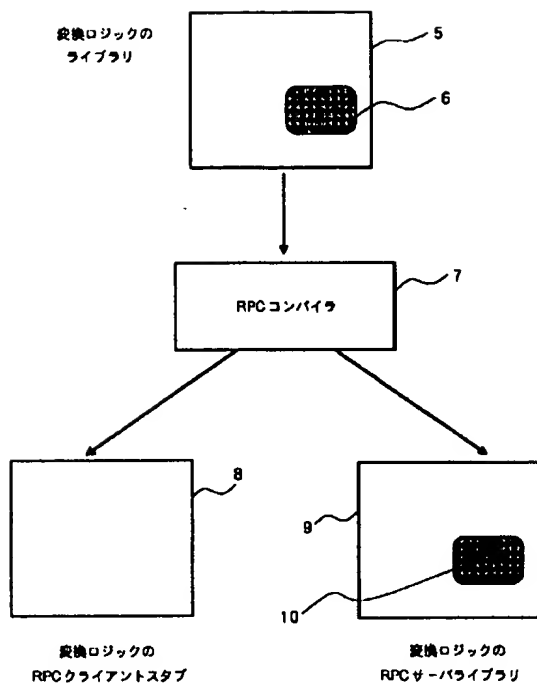
【図1】



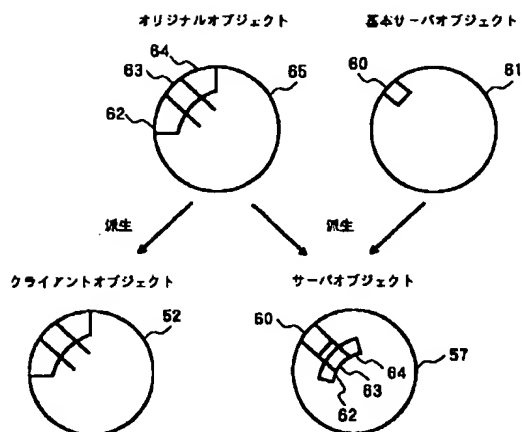
【図2】



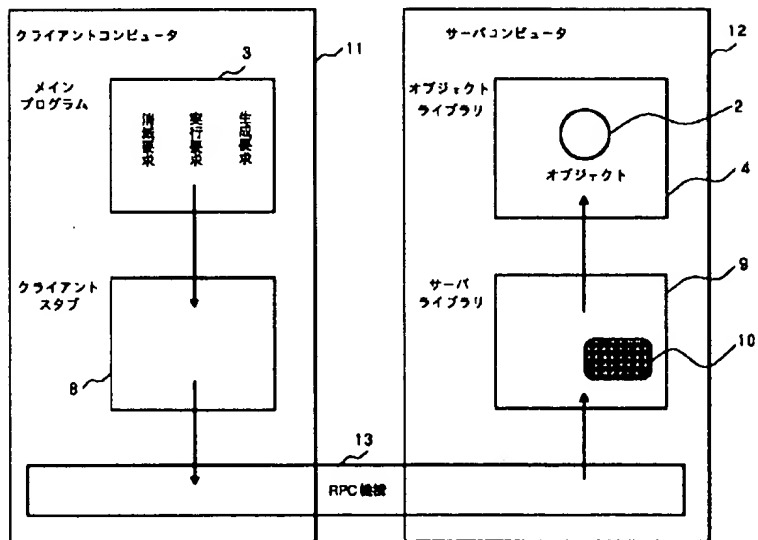
【図3】



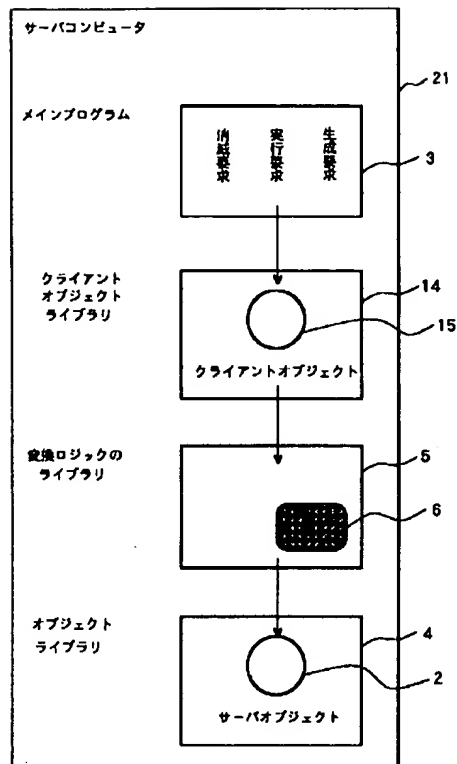
【図8】



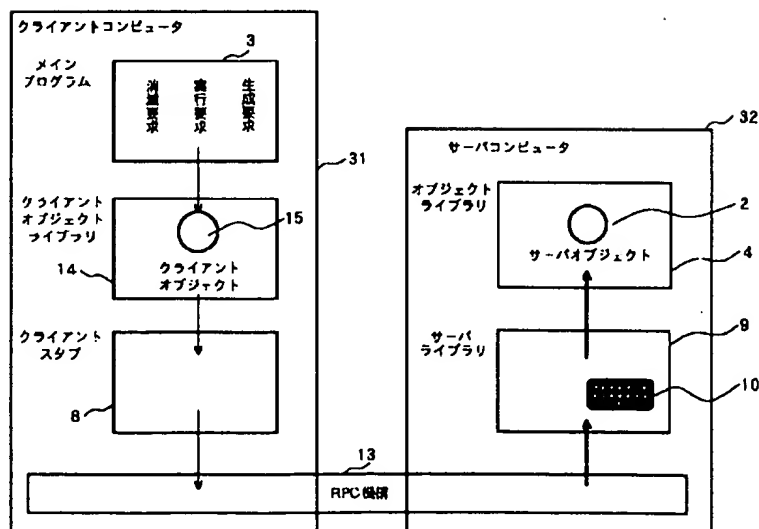
【図4】



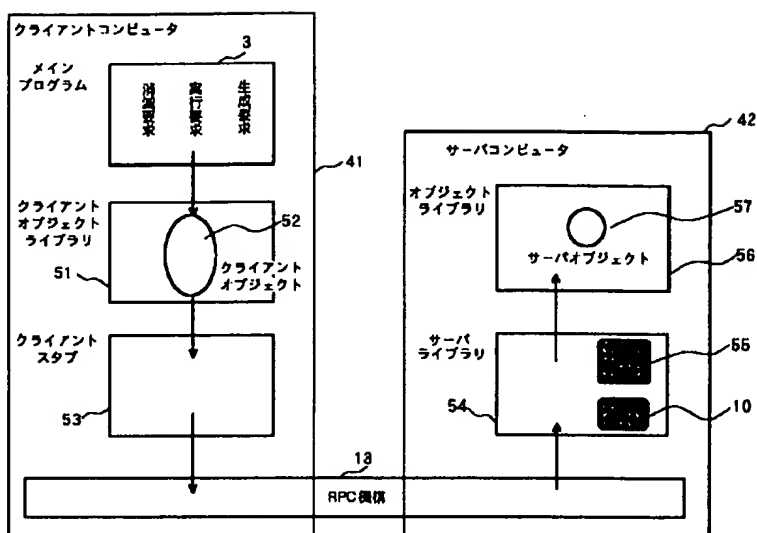
【図5】



【図6】



【図7】



PTO 04-1912

Japanese Patent

Document No. H10-275082

REMOTE OBJECT ACCESS SYSTEM AND METHOD

[Remoto Obujekuto Akusesu Shisutemu Oyobi Hoho]

Takashi Yamada

UNITED STATES PATENT AND TRADEMARK OFFICE

Washington, D.C.

February 2004

Translated by: Schreiber Translations, Inc.

Country : Japan

Document No. : H10-275082

Document Type : Kokai

Language : Japanese

Inventor : Takashi Yamada

Applicant : NTT Data Communications Co., Ltd.

IPC : G 06 F 9/44
9/46
13/00

Application Date : February 2, 1998

Publication Date : October 13, 1998

Foreign Language Title : Remoto Obujekuto Akusesu Shisutemu
Oyobi Hoho

English Title : REMOTE OBJECT ACCESS SYSTEM AND
METHOD

(54) Title of the invention

Remote object access system and method

(57) Summary (amendments included)

Objective: Attempts are made, by using a remote procedure call (RPC), to generate an object in a dynamic fashion on a computer separated via a network and to access the same.

Solution mechanism: The client computer (11) possesses the main program (3) and the transform logic client stub (8). The server computer (12) possesses the transform logic server library (9). The transform logic server library (9) generates, in response to a generation request issued from the main program (3), the object (2), assigns a unique ID to said object, stores said object ID within the correspondence table (10) together with a pointer, and then returns said object ID to the main program (3). The main program (3) subsequently issues, by designating said object ID, a processing request or deletion request for said object (2). The transform logic server library (9) acquires the pointer corresponding to said object ID from the correspondence table (10) and then processes the object (2) pointed by said pointer.

Patent Claims

/2

Claim 1

¹ Numbers in the margin indicate pagination in the foreign text.

5/12/78
11A
sheet

A remote object access system with the following characteristics: In a remote object access system which permits, via the remote procedure call (RPC), a client computer to generate an object on a server computer and to access said object,

The aforementioned client computer possesses

An RPC client stub which issues, in response to individual requests for generating and accessing objects arisen on the aforementioned client computer, actions for calling individual functions for generating and accessing objects to the aforementioned server computer via the aforementioned remote procedure call, whereas

The aforementioned server computer possesses

An RPC server library which possesses functions for generating and accessing the aforementioned objects, which generates server objects on the aforementioned server computer in response to actions for calling individual functions received from the aforementioned RPC client stub, and which then executes the access to said server objects, whereas

The RPC client stub and the aforementioned RPC server library exchange, via the aforementioned remote procedure call, unique instance IDs assigned to the respective server objects, whereas

The aforementioned RPC server library possesses a mechanism for transforming the instance IDs of the respective server objects into pointers.

Claim 2

A remote object access method with the following characteristics: In a method which permits a client computer to generate, via the remote procedure call (RPC), an object on a server computer and to access said object,

A process whereby the aforementioned client computer serves the function of an RPC client stub which issues, in response to individual requests for generating and accessing objects generated on the aforementioned client computer, actions for calling individual functions for generating and accessing said objects to the aforementioned server computer via the aforementioned remote procedure call,

A process whereby the aforementioned server computer serves the function of an RPC server library which possesses functions for generating and accessing the aforementioned objects, which generates, in response to actions for calling the individual functions received from the aforementioned RPC client stub, server objects on the aforementioned server computer, and which executes the access to said server objects,

A process whereby unique instance IDs assigned to the respective server objects are exchanged, via the aforementioned remote procedure call, between the aforementioned RPC client stub and the aforementioned RPC server library, and

A process whereby the aforementioned RPC server library transforms the instance IDs of the respective server objects into pointers

Are included.

Claim 3

A remote object access system with the following characteristics: In a remote object access system which permits, via the remote procedure call (RPC), a client computer to generate an object on a server computer and to access said object,

The aforementioned client computer

Possesses an RPC client stub which issues, in response to individual requests for generating and accessing objects arisen on the aforementioned client computer, actions for calling individual functions for generating and accessing objects to the aforementioned server computer via the aforementioned remote procedure call, whereas

The aforementioned server computer

Possesses an RPC server library which possesses functions for generating and accessing the aforementioned objects, which generates server objects on the aforementioned server computer in response to actions for calling individual functions received from the aforementioned RPC client stub, and which then executes the access to said server objects, whereas

The aforementioned RPC server library is designed to assign unique instance IDs to the generated server objects, to retain a correspondence table between the instance IDs of the respective server objects and pointers, and to return, to the aforementioned RPC client stub, the instance IDs assigned to the aforementioned generated server objects, whereas

The aforementioned RPC client stub incorporates, into the aforementioned function call, the returned instance IDs of the respective server objects at the time of issuing an action for calling the functions for accessing the respective server objects.

Claim 4

A remote object access system with the following characteristics: In the system specified in Claim 3,

The aforementioned client computer additionally possesses a client object which becomes generated in response to a request for generating the aforementioned object, whereas

The aforementioned client object

Is designed not only

(1): To command, at the time of its generation, the aforementioned RPC client stub to issue an action for calling the function for generating the aforementioned object and to cache the instance ID returned from the aforementioned RPC client stub as a result of the execution of said generation function but also

(2): To command, in response to a request for accessing the aforementioned object, the aforementioned RPC client stub to issue, by using the aforementioned instance ID being cached, a function for gaining the aforementioned access.

Claim 5

A remote object access system with the following characteristics: In the system specified in Claim 4, the aforementioned client object is designed to additionally possess an object type-specific ID which shows the type of said object and

to command, at the time of the generation of said client object, the aforementioned RPC client stub to issue, by using the aforementioned object type-specific ID, a function calling action for generating the aforementioned object, whereas /3

The aforementioned RPC client stub is designed, at the time of issuing a function calling action for generating the aforementioned object, to incorporate the aforementioned object type-specific ID into the aforementioned function call, whereas

The aforementioned RPC server library is designed to additionally possess a correspondence table between object types and object type-specific IDs and to generate, in response to the function calling action for generating the aforementioned object, a server object which is formatted by a method supported by the original object that belongs to the object type corresponding to the aforementioned object type-specific ID.

Claim 6

A remote object access system with the following characteristics: In the system specified in Claim 5,

The aforementioned client object is designed to be branched from the aforementioned original object and to accept a request for the method supported by the aforementioned original object, whereas

The aforementioned server object is designed to be branched not only from the aforementioned original object and a certain foundational server object and to accept a request for the method

supported by the aforementioned foundational server object, whereas

The aforementioned client object is additionally designed, in response to the request for the method supported by the aforementioned original object, to designate a routine type assigned preliminarily to said method, and to command the aforementioned RPC client stub to issue an action for calling the routine request function specific to the method supported by the aforementioned foundational object, whereas

The aforementioned RPC server library is additionally designed, in response to the action for calling the processing request function specific to the method supported by the aforementioned foundational object, to designate the aforementioned routine type and to issue, to the aforementioned server object, a request for the method supported by the aforementioned foundational server object, whereas

The aforementioned server object is additionally designed, in response to the request for the method supported by the aforementioned foundational server object, to execute, among the methods supported by the original object, the method corresponding to the aforementioned routine type.

Claim 7

[A remote object access system with the following characteristics:] In a remote object access system which permits, via the remote procedure call (RPC), a client computer to generate an object on a server computer and to access said object,

The aforementioned client computer

Possesses an RPC client stub which issues, in response to individual requests for generating and accessing objects arisen on the aforementioned client computer, actions for calling individual functions for generating and accessing objects to the aforementioned server computer via the aforementioned remote procedure call, whereas

The aforementioned server computer

Possesses an RPC server library which possesses functions for generating and accessing the aforementioned objects, which generates server objects on the aforementioned server computer in response to actions for calling individual functions received from the aforementioned RPC client stub, and which then executes the access to said server objects, whereas

The RPC client stub and the aforementioned RPC server library exchange, via the aforementioned remote procedure call, unique instance IDs assigned to the respective server objects, whereas

The aforementioned RPC server library is a computer-decodable recording medium which is loaded with a program for invoking the computer function of the aforementioned client computer within a remote object access system in possession of a mechanism for transforming the instance IDs of the respective server objects into pointers.

Claim 8

[A remote object access system with the following characteristics:] In a remote object access system which permits,

via the remote procedure call (RPC), a client computer to generate an object on a server computer and to access said object,

The aforementioned client computer

Possesses an RPC client stub which issues, in response to individual requests for generating and accessing objects arisen on the aforementioned client computer, actions for calling individual functions for generating and accessing objects to the aforementioned server computer via the aforementioned remote procedure call, whereas

The aforementioned server computer

Possesses an RPC server library which possesses functions for generating and accessing the aforementioned objects, which generates server objects on the aforementioned server computer in response to actions for calling individual functions received from the aforementioned RPC client stub, and which then executes the access to said server objects, whereas

The RPC client stub and the aforementioned RPC server library exchange, via the aforementioned remote procedure call, unique instance IDs assigned to the respective server objects, whereas

The aforementioned RPC server library is a computer-decodable recording medium which is loaded with a program for invoking the computer function of the aforementioned server computer within a remote object access system in possession of a mechanism for transforming the instance IDs of the respective server objects into pointers.

Claim 9

[A remote object access system with the following characteristics:] In a remote object access system which permits, via the remote procedure call (RPC), a client computer to generate an object on a server computer and to access said object,

/4

The aforementioned client computer possesses

An RPC client stub which issues, in response to individual requests for generating and accessing objects arisen on the aforementioned client computer, actions for calling individual functions for generating and accessing objects to the aforementioned server computer via the aforementioned remote procedure call, whereas

The aforementioned server computer

Possesses an RPC server library which possesses functions for generating and accessing the aforementioned objects, which generates server objects on the aforementioned server computer in response to actions for calling individual functions received from the aforementioned RPC client stub, and which then executes the access to said server objects, whereas

The aforementioned RPC server library is designed to assign unique instance IDs to the generated server objects, to retain a correspondence table between the instance IDs of the respective server objects and pointers, and to return, to the aforementioned RPC client stub, the instance IDs assigned to the aforementioned generated server objects, whereas

The aforementioned RPC client stub is a computer-decodable recording medium which is loaded with a program for invoking the computer function of the aforementioned client computer within a remote object access system designed, at the time of issuing an action for calling the functions for accessing the respective server objects, to incorporate, into the aforementioned function call, the returned instance IDs of the respective server objects.

Claim 10

[A remote object access system with the following characteristics:] In a remote object access system which permits, via the remote procedure call (RPC), a client computer to generate an object on a server computer and to access said object,

The aforementioned client computer

Possesses an RPC client stub which issues, in response to individual requests for generating and accessing objects arisen on the aforementioned client computer, actions for calling individual functions for generating and accessing objects to the aforementioned server computer via the aforementioned remote procedure call, whereas

The aforementioned server computer

Possesses an RPC server library which possesses functions for generating and accessing the aforementioned objects, which generates server objects on the aforementioned server computer in response to actions for calling individual functions received from the aforementioned RPC client stub, and which then executes the access to said server objects, whereas

The aforementioned RPC server library is designed to assign unique instance IDs to the generated server objects, to retain a correspondence table between the instance IDs of the respective server objects and pointers, and to return, to the aforementioned RPC client stub, the instance IDs assigned to the aforementioned generated server objects, whereas

The aforementioned RPC client stub is a computer-decodable recording medium which is loaded with a program for invoking the computer function of the aforementioned server computer within a remote object access system designed to incorporate, into the aforementioned function calling action, the returned instance IDs of the respective server objects at the time of issuing an action for calling the functions for accessing the respective server objects.

Detailed explanation of the invention

[0001]

(Technical fields to which the invention belongs)

The present invention concerns a remote object access system for operating an object on another computer separated via a network by using the remote procedure call ([transliterated in Japanese]; hereafter abbreviated as the "RPC").

[0002]

Incidentally, the RPC is divided, in terms of loading morphologies, into a format whereby individual processing requests

received from a network are processed within mutually independent memory spaces by initializing processes in request-specific fashions and a format wherein the same are processed within a common memory space by using a thread or cueing logic, whereas the present method is applicable to the latter.

[0003]

(Prior art)

The RPC is being used for exchanging procedural programming calls between computers separated via a network. The RPC represents a technology capable of calling procedures on a computer separated via a network as if they were procedures on a local computer. In a case where a given computer calls for the procedures of another computer by using the RPC, said procedures become executed by said "another" computer, and upon the completion of the execution of said procedures, the results are returned to the latter from the former.

[0004]

(Problems to be solved by the invention)

Coincidentally, in an object-oriented programming context, operations for objects such as the generations of objects, requests for processing the same, and/or deleting the same are executed in place of procedural calls prevailing in the procedural programming context. With regard to such object operations in the object-oriented programming context, too, it is desirable for an

object on a second computer to be handled as if it were an object on a local computer between such computers separated via a network.

[0005]

The following problem, however, is observed in a case where an attempt is made to execute, in relation to an object on a second computer separated via a network, procedures similar to those of object operations within a local computer.

[0006]

Figure 1 shows a normal program model in a case where the object (2) is operated within the local computer (1). According to this model, the main program (3) engages in the following operations.

/5

[0007]

1. An object generation request is issued, as a result of which the pointer for the object (2) generated within the object library (4) is acquired.

[0008]

2. Subsequently, a request for executing a desired routine on the object (2) is issued by using the acquired pointer.

[0009]

3. In a case where the object (2) has become unnecessary, a request for deleting the object (2) is issued by using the acquired pointer.

[0010]

Thus, the normal program model within the local computer is designed to execute the operation of an object by using the generated object pointer.

[0011]

In a case where an attempt is made to execute such an object operation in relation to an object on another computer by using the RPC of the prior art, it is inevitable for the operation request to be transformed into a simple procedural language interface provided by the RPC. For this reason, the generation of the object, requests for processing the same, and the deletion of the same are each defined as procedures (functions) of the procedural language, and the aforementioned pointers must be designated as index numbers of such functions. In a case where an object is generated on another computer separated via a network, however, the memory space is variable, and therefore, even if the pointer for said object is acquired, there is absolutely no guarantee that the same pointer can be used for designating the same object during subsequent actions for calling other functions. In other words, it may be possible, during the progress of the action for calling a singular function, to generate an object, to request a routine for processing said object, and to delete the same, but the format for exchanging a pointer that utilizes the RPC of the prior art is incapable of generating an object based on a single function calling operation, of issuing a request for processing said object by calling another function, and of deleting said object by calling still another function. In such a

case, it is impossible to fully take advantage of the unique advantage of the object-oriented language, namely adventitious generations and accesses of objects in dynamic fashions.

[0012]

The objective of the present invention is therefore to provide a remote object access system which is capable of generating an object on a computer separated via a network by using the RPC in a dynamic fashion and of accessing the same.

[0013]

(Mechanism for solving the problems)

The system pertaining to the first aspect of the present invention is a system which permits, via the RPC, a client computer to generate an object on a server computer and to access said object (i.e., to request the execution of a routine and/or deletion), whereas in the context of generating an object, the client computer issues a request for generating a procedural language interface, in response to which an object is generated by the server computer, and after a unique ID has been assigned to said object, said ID is returned, as a response, to the client computer. The server computer subsequently manages the correspondence relationship between the ID of the object and its pointer. The client computer then issues an object access request for the procedural language interface by designating the ID of the object, in response to which the server computer accesses the object designated by the pointer corresponding to said ID.

[0014]

In addition to being endowed with the aforementioned constitutional attributes, the system pertaining to the second aspect of the present invention is characterized by the fact that the client computer possesses a hypothetical object (referred to as the "client object") corresponding to the object within the server computer (referred to as the "server object"). The client object supports a method which bears a name identical to that of the corresponding server object, whereas during its generation and deletion phases, a request for generating the procedural language interface and a request for deletion are issued, whereas in a case where a request for the supported method has become received, it is transformed into a corresponding request for the procedural language interface. The server computer may, in response to the request issued from the client object, generate and/or access the corresponding server objects. The system pertaining to the third aspect of the present invention is, in addition to being endowed with the constitutional attributes of the system pertaining to the second aspect, further in possession of a mechanism for designating the type (class) of a generated object.

[0015]

Incidentally, the programs of the respective computers that constitute the system of the present invention can be installed or loaded into said computers via various media such as disc-type storage units, semiconductor memories, communications lines, etc.

[0016]

(Application embodiments of the invention)

In the following, application embodiments of the present invention will be explained with reference to attached figures. Incidentally, in subsequent explanations, a computer which issues a request for operations on objects (generation, routine execution, and/or deletion) will be referred to as the "client computer," whereas a computer which executes the operations on objects upon the receptions of such requests as the "server computer."

[0017]

The remote object access system of the first application embodiment of the present invention will be explained with reference to Figures 2 through 4.

[0018]

This application embodiment can be outlined as follows. In other words, the server computer assigns, at the time of the generation of an object in response to a generation request issued from the client computer, a unique ID to said object, retains a correspondence table between the pointers and IDs for such objects, and to return, as a response, the corresponding ID to the client computer. The client computer retains the ID received from the server computer, whereas subsequent operations on the object on the server computer are executed based on function calling actions by designating the ID of the object.

/6

[0019]

Figure 2 shows a normal program model which, within the local computer (1), lays the foundation for building the remote object access system of the present application embodiment.

[0020]

As the comparison with the program of the prior art shown in Figure 1 clearly indicates, the program model shown in Figure 2 additionally possesses the transform logic library (5) between the main program (3) and the object library (4). This transform logic library (5) represents a collection of procedures (functions) for object operations, namely the generations of objects, requests for routine executions, and deletions, whereas in a case where an object generation function call is received from the main program (3), the object (2) is generated within the object library (4), and at the same time, a unique ID is assigned to said object (2) for identification purposes, whereas the pointer for the object (2) acquired from the object library (4) and the ID assigned to the object (2) are stored, retained, and managed in the correspondence table (6).

[0021]

In the following, actions for generating and accessing objects by using this program model will be explained.

[0022]

1. Generation of an object

The main program (3) issues an object generation function call to the transform logic library. The transform logic library (5) proceeds to generate the object (2) within the object library (4), to determine a unique ID, to store the acquired pointer and ID in the correspondence table (6), and then to return said ID to the main program (3) [sic: One extraneous "ID" out of context]. The main program (3) acquires, from the transform logic library (5), the ID for the generated object (2).

[0023]

2. Execution of an object routine

The main program (3) issues, by setting the acquired ID as an index number, a routine execution request function call for the object (2) to the transform logic library (5). The transform logic library (5) acquires the pointer corresponding to the ID from the correspondence table (6) and then issues, by using said pointer, a routine execution request to the object (2).

[0024]

3. Deletion of the object

The main program (3) issues, after the object (2) has become unnecessary, an object deletion function call to the transform logic library (5) by setting the acquired ID as an index number. The transform logic library (5) acquires the pointer corresponding to the ID from the correspondence table (6) and then issues, by using said pointer, a request for deleting the object (2).

[0025]

Figure 3 is a diagram which shows the manners by which program units mounted respectively on the client computer and server computer of the system of the present application embodiment are generated based on the normal program model transform logic library (5) shown in Figure 2.

[0026]

In a case where the transform logic library (5) shown in Figure 2 is inputted into the conventionally-known RPC compiler (7), the RPC client stub for the transform logic (hereafter referred to simply as the "client stub") (8) and the RPC server library for the transform logic (hereafter referred to simply as the "server library") (9) become automatically generated. The client stub (8) and server library (9) hereby serve, in a state where they are mutually being coupled via an RPC mechanism, a collective function similar to that served by the transform logic library (5) shown in Figure 2. The transform logic client stub (8) provides an interface identical to that for the transform logic library (5) to the main program (3) and is endowed with a function of transforming this interface into the procedural interface for the RPC. This server library (9) retains and manages the correspondence table (10), which is equivalent to the correspondence table (6) within the transform logic library (5), and is endowed with the essential functions of the transform logic library (5), namely functions of, in response to object operation requests arriving from the client stub (8) via the RPC mechanism,

actually generating an object, issuing routine execution requests to said objects, and deleting said objects.

[0027]

Figure 4 shows a program model of the system of the present application embodiment constituted by computers loaded respectively with the aforementioned client stub (8) and transform logic server library (9).

[0028]

As Figure 4 indicates, the client computer (11) and the server computer (12) exist apart from one another via a network, whereas both can communicate with one another via the RPC mechanism (13). The client computer (11) possesses the main program (3) and the client stub (8) shown in Figure 3. The server computer (12) possesses the object library (4) and the server library (9) shown in figure 3. The server library (9) possesses, as has been discussed earlier, the correspondence table (10) for the pointers for the respective objects (2) within the object library (4) and their IDs. As has been mentioned earlier, the client stub (8) and server library (9), which are being mutually coupled via the RPC mechanism (13), serves a collective function similar to that of the transform logic library (5) of the normal program model shown in Figure 2. Since it is not the pointer for the object but its ID that is exchanged via the RPC mechanism (13), no problems arise from the mutually different memory spaces of the client computer (11) and the server computer (12). The client computer (11) is therefore capable of operating on the

object (2) on the server computer (12), which is separated via the network.

[0029]

Next, the object operation in the system of Figure 4 will be explained.

[0030]

1. Generation of an object

In a case where the main program (3) of the client computer (11) has issued an object generation function call to the client stub (8), said client stub (8) transmits this function call to the server library (9) via the RPC mechanism (13). The server library (9) generates, in response to this function call, the object (2) within the object library (4), acquires the pointer for the object (2) from the object library (4), assigns a unique ID to the object (2), stores said pointer and ID in the /7
correspondence table (10), and then returns said ID to the client stub (8) via the RPC mechanism (13). This ID is relayed to the main program (3) from the client stub (8), and said ID is retained by the main program (3).

[0031]

2. Execution of an object routine

The main program (3) issues, by setting the acquired ID as an index number, a routine execution request function call for the object (2) to the client stub (8). The client stub (8) transmits

said function call to the server library (9). The server library (9) acquires, in response to said call, the pointer corresponding to said ID from the correspondence table (10) and then issues a routine execution request for the object (2) specified by said pointer.

[0032]

3. Deletion of the object

The main program (3) issues, after the object (2) has become unnecessary, an object deletion function call to the client stub (8) by setting the acquired ID as an index number. The client stub (8) transmits said function call to the transform logic server library (9). The server library (9) acquires a pointer corresponding to said ID from the correspondence table (10) and then issues an deletion request to the object (2) specified by said pointer.

[0033]

In the following, a loading example of the transform logic library (5) of the system of the present application example will be shown. A loading example pertaining to the class of "Object A" of C++, which is an object-oriented language, will be explained. In this context, Object A is presumed to be endowed with three methods, namely "Action 1," "Action 2," and "Action 3." Incidentally, the generation, deletion, and methods of Object A are presumed to possess no parameters for the sake of simplicity.

[0034]

In a case where the transform logic library (5) is loaded, the following five functions are prepared as object operation functions by using the C language, which instantiates a procedural language.

[0035]

1. Object generation function: "Object A Create (ID, ret)"

This is a function for generating an instance (object) of the Object A class. This function stores, upon the generation of an instance of the Object A class, the pointer for said instance and an ID assigned to the same in the correspondence table (10) and returns said ID and generation result ret to the call origination.

[0036]

2. Object deletion function: "Object A Delete (ID, ret)"

This is a function for deleting the instance (object) of the Object A class. This function acquires, based on the relayed ID, the pointer for the corresponding instance from the correspondence table (10), deletes the instance specified by said pointer, deletes the entry from the correspondence table (10), and returns the result ret to the call origination.

[0037]

3. Action 1 request function: "Object A Action 1 (ID, ret)"

This is a function for requesting the instance of Action 1 as a method. This function acquires, based on the relayed ID, the

pointer for the corresponding instance from the correspondence table (10), issues Action 1 to the instance specified by said pointer, and returns the result of said Action 1, as the result ret, to the call origination.

[0038]

4. Action 2 request function: "Object A Action 2 (ID, ret)"

5. Action 3 request function: "Object A Action 3 (ID, ret)"

The procedures are identical to those for Object A Action 1 (ID, ret) except that Action 2 and Action 3 are issued respectively as methods.

[0039]

In a case where the foregoing functions of C are loaded as the transform logic library (5) shown in Figure 2 and where the client stub (8) and the server library (9) are, as Figure 4 shows, loaded respectively into the computers (11) and (12) on the network under the pervasion of the resulting compilation shown in Figure 3, it becomes possible, in response to a request issued from the client computer (11), to generate an instance of the Object A class on the server computer (12), to request the same of Action 1, Action 2, or Action 3, and/or to delete the same.

[0041]

Since it is possible to generate an object on a remote computer, to request a routine for the same, and to delete the same based on the first application embodiment shown above, the following effects can be achieved.

[0041]

(1): As a result of the routine request for the object, it becomes possible to retain, on a remote machine, information generated within the object.

[0042]

(2): The information generated within the object can be retrieved by issuing an independent routine request.

[0043]

(3): The object on the remote computer can be shared by multiple computers.

[0044]

Next, the remote object access system of the second application embodiment of the present invention will be explained with reference to Figures 5 and 6.

[0045]

This application embodiment can be outlined as follows. In correspondence to an object on a server computer (referred to as the "server object"), an object which possesses an interface identical to that of the former (referred to as the "client object") becomes generated on a client computer. The server object hereby represents the "true" object, whereas the client object may, on the other hand, be construed as a "hypothetical" object the interface of which alone is identical to that of the server object. In a case where a main program within the client computer operates on

/8

the client object, a corresponding operation on the server object becomes executed on the server computer. As far as the first application embodiment is concerned, the object operation originating from the main program must be executed by using a procedural language such as the C language, whereas an object-oriented language can be used in the second application example.

[0046]

Figure 5 shows a normal main program within a local computer that lays the foundation for building the remote object access system of the present application embodiment.

[0047]

The local computer (21) possesses the main program (3), the transform logic library (5), and the object library (4), as in the case of the model shown in Figure 2, as well as the client object library (14) between the main program (3) and the transform logic library (5). This client object library (14) possesses the client objects (15) corresponding to the respective server objects (2).

[0048]

In the following, the object operations according to the aforementioned model will be explained.

[0049]

1. Generation of an object

The main program (3) generates the client object (15) within the client object library (14). The client object (15) issues, in the midst of the generation routine, an object generation function

call to the transform logic library (5). The transform logic library (5) generates, in response to this call, the server object (2), acquires the pointer for said object (2), assigns a unique ID to said object (2), stores the pointer and ID in the correspondence table (6), and then returns the ID to the client object (15). The client object (15) retains, as an attribute, said ID.

2. Execution of object routine

The main program (3) requests the client object (15) to execute the routine. The client object (15) issues, by setting, as an index number, the ID being retained as an attribute, a routine execution request function call for the object (2) to the transform logic library (5). The transform logic library (5) acquires the pointer corresponding to the ID from the correspondence table (6) and then issues a routine execution request to the object (2) specified by said pointer.

3. Deletion of the object

The main program (3) deletes the client object (15) that has become unnecessary. The client object (15) issues, in the midst of the deletion routine, an object deletion function call to the transform logic library (5) by setting, as an index number, the ID being retained as an attribute. The transform logic library (5) acquires the pointer corresponding to the ID from the

correspondence table (6) and then issues a deletion request to the object (2) specified by said pointer.

[0052]

Figure 6 shows a program model for the system of the present application example built based on the foundation of the model shown in Figure 5.

[0053]

As the figure indicates, the client computer (31) possesses the main program (3), the client object library (14), and the client stub (8). The server computer (32) possesses the server library (9) and the object library (4). The client stub (8) and the server library (9) are, as Figure 3 shows, generated by compiling the transform logic library (5) by using a conventionally-known RPC compiler.

[0054]

The object operation for the system of the present application example is executed according to the following procedures.

[0055]

1. Generation of an object

The main program (3) generates the client object (15) within the client object library (14). The client object (15) generates, in the midst of the generation routine, an object generation function call to the client stub (8). The client stub (18) [sic: Presumably "(8)"] transmits, via the RPC mechanism (13), said function call to the server library (19) [sic: Presumably "(9)"].

The server library (9) generates, in response to said call, the object (2), acquires the pointer for the object (2), assigns a unique ID to the object (2), stores said pointer and ID in the correspondence table (10), and returns the ID to the transform logic client stub (8). The client stub (8) relays said ID to the client object (15), whereas the client object (15) retains said ID as an attribute.

[0056]

2. Execution of object routine

The main program (3) requests the client object (15) to execute a routine. The client object (15) issues, by setting, as an index number, the ID being retained as an attribute, a routine execution request function call to the server library (9) via the client stub (8). The server library (9) acquires the pointer corresponding to the ID from the correspondence table (10) and then issues a routine execution request to the object (2) specified by said pointer.

[0057]

2. [sic: Presumably "3."] Deletion of the object

The main program (3) deletes the client object (15) that has become unnecessary. The client object (15) issues, in the midst of the deletion routine, an object deletion function call to the server library (9) via the RPC client stub (8) by setting, as an index number, the ID being retained as an attribute. The server

library (9) acquires the pointer corresponding to the ID from the correspondence table (10) and then issues a deletion request to the object (2) specified by said pointer.

[0058]

In the following, a loading example of the system of the present application embodiment will be shown.

[0059]

/9

This loading example of the present application embodiment will be explained by designating "Object A" of C++ as a server object class, as in the case of the loading example of the first application embodiment, and by assuming that said "Object A" possesses, as methods, "Action 1," "Action 2," and "Action 3."

[0060]

First, five functions of the C language, namely "Object A Create (ID, ret)," "Object A Delete (ID, ret)," "Object A Action 1 (ID, ret)," "Object A Action 2 (ID, ret)," and "Object A Action 3 (ID, ret)," are prepared as the transform logic library (5), as in the case of the loading example of the first application embodiment.

[0061]

Next, an Object B class is prepared as a client object class corresponding to the Object A class of the server object. Incidentally, the name of Object B may be identical to the name of Object A so long as Object A is not used within the main program.

A method the name of which is identical to that of the method supported by Object A is prepared as this Object B, and the respective methods are loaded for enabling the calls of the aforementioned C functions. These methods are characterized as follows.

[0062]

1. Constructor (generation routine): "Object B : : Object B"

This calls the generation function Object A Create (ID, ret) and stores the returned ID in the system interior.

[0063]

2. Destructor (deletion routine): "Object B : : ~ Object B"

This issues, by using the ID being retained in the system interior, the deletion function call Object A Delete (ID, ret).

[0064]

3. "Object B : : Action 1"

The Action 1 request function call Object A Action 1 (ID, ret) is issued by using the ID being retained in the system interior.

4. "Object B : : Action 2"

5. "Object B : : Action 3"

These are identical to Action 1 except that Object A Action 2 (ID, ret) and Object A Action 3 (ID, ret) are respectively issued as function calls.

[0066]

The following advantages are achieved according to the second application embodiment discussed above.

[0067]

(1): Since a client object can be mechanically generated from an object scheduled to be configured on a network, a network program can be generated with ease.

[0068]

(2): An object on the network can be controlled with a pointer according to procedures similar to those for an object within a local computer.

(3): An object management logic within a local computer can also be applied to an object on a remote computer.

(4): As an expanded version of the format of the present application embodiment, an effect of dispersing the load between the server computer and client computer can be realized by partially loading a client object in the course of a real routine instead of construing it as an utterly hypothetical one.

[0071]

Next, the remote object access system of the third application embodiment of the present invention will be explained with reference to Figures 7 and 8.

[0072]

This application embodiment is provided by adding, to the second application embodiment discussed above, a mechanism capable of arbitrarily designating the type (class) of an object selected as an operation target. In a case where one wishes to change the class of an object that prevails as an operation target, it is necessary to reconstruct a transform logic library according to the second application embodiment, whereas such a measure is unnecessary according to the present application embodiment.

[0073]

Figure 7 shows a program model for the remote object access system of the present application embodiment.

[0074]

The client computer (41) possesses the main program (3), the object library (51), and the RPC client stub for the transform logic (hereafter referred to simply as the "client stub") (53). The object library (51) retains the client object (52), an identical assigned to the former (hereafter referred to as the "instance ID"), and a type-specific ID which shows the type (class) of said object.

[0075]

The server computer (42) possesses the RPC server library for the transform logic (hereafter referred to simply as the "server library") (54) and the object library (56). The object library (56) retains the server object (57). The client object (52) corresponds to the server object (57) at a 1 : 1 ratio, and their

concrete constitutions will be explained on a later occasion. The server library (54) retains the correspondence table (10) between the pointer & instance ID of the server object (57) and the correspondence table (55) between the object type-specific ID & type (class) of the server object (57).

[0076]

The client stub (53) and the server library (54) are generated by compiling the following transform logic library by using a conventionally-known RPC compiler. This transform logic library possesses the following three functions.

[0077]

(1): Object generation function

This function acknowledges, with reference to the table (55) based on the object type-specific ID included in the call received from the client object (52), the type (class) corresponding to said object type-specific ID, generates the object (57) of the corresponding type in the object /10 library (56), assigns a unique ID to said object (57), stores the instance ID and pointer for said object (57) in the table (10), and then returns said instance ID to the client object (52). Incidentally, the server object (57) hereby generated is designed to be branched from a foundational server object, as will be discussed on a later occasion.

[0078]

(2): Object deletion function

This function acquires, with reference to the table (10) based on the instance ID included in the call received from the client object (52), the pointer corresponding to said instance ID and then deletes the server object (57) specified by said pointer.

[0079]

(3): Routine request function

This function acquires, with reference to the table (10) based on the instance ID included in the call received from the client object (52), the pointer corresponding to said instance ID and then issues, to the server object (27) [sic: Presumably "(57)"] specified by said pointer, a method supported by the foundational server object, which will be discussed on a later occasion. This method request includes a routine type designation which, too, will be discussed on a later occasion.

[0080]

Figure 8 shows the respective constitutions of the client object (52) and the server object (57). _

[0081]

In Figure 8, the foundational server object (61) is a preliminarily prepared server object of a given class, and it supports the singular method (60). The original object (65) is an object which the main program (3) intends to generate, and in this embodiment, three methods (62), (63), and (64) are presumed to be supported.

[0082]

As the figure indicates, the client object (52) is designed to be branched from the original object (25) [sic: Presumably "(65)"], and it possesses, as an attribute, the object type-specific ID assigned to the original object (25) [sic] at the time of system construction. This client object (52), which is illustrated to be morphologically identical to the original object (65) in the figure, provides the original object (65) and an interface to the main program (3). This client object (52), which may be construed as a hypothetical version of the original object (65), engages in the following actions.

[0083]

<1>: It executes a generation routine in response to a generation request received from the main program (3), whereas during said generation routine, it issues the aforementioned object generation function call to the aforementioned transform logic library by setting, as an index number, the object type-specific ID possessed as an attribute, and it then retains, as an attribute, the instance ID returned from the transform logic library.

[0084]

<2>: It issues, upon the reception of a request for the method (62), (63), or (64) supported by the original object (65) from the main program (3), the aforementioned routine request function call to the transform logic library by setting, as index numbers, the instance ID possessed as an attribute and the routine

type assigned to the requested method (62), (63), or (64). Incidentally, the respective routine types assigned to the methods (62), (63), and (64) are assigned between the client object (52) and the server object (57) at the time of system construction.

[0085]

<3>: It executes, in response to a deletion request received from the main program, a deletion routine and issues, during said deletion routine, the aforementioned object deletion function call by setting, as an index number, the instance ID possessed as an attribute.

[0086]

As Figure 8 indicates, the server object (57) is designed to be branched from a pair of objects, namely the original object (65) and the foundational server object (61), whereas it possesses, as an interface with the outside, the method (60) supported by the foundational server object (61), whereas it additionally possesses, in the interior of said method (60) shielded from the outside, the methods (62), (63), and (64) supported by the original object (65), whereas the correspondences of these methods (62), (63), and (64) with the aforementioned routine types are established.

[0087]

This server object (57) executes, upon the reception of a request for the method (60) supported by the foundational server object (60) [sic: Presumably "(61)"] from the transform logic

library, the method (62), (63), or (64) corresponding to the routine type included in said request.

[0088]

It becomes possible, based on the foregoing model, to generate, according to operative procedures identical to those for generating an object (original object) of a certain type (class) within a host computer itself, the true object of the corresponding class [server object (57)] on the network, and to induce the server object (57) on the network to execute, according to procedures identical to those for requesting a desired method in relation to a desired object within the host computer itself, the desired method.

[0089]

The following advantages are achieved based on the present application embodiment.

[0090]

(1): A server object and a client object can be generated mechanically from an object which one wishes to configure on a network, based on which a network program can be generated with ease.

[0091]

(2): An object on the network can be controlled with a pointer in a manner similar to that for an object within a computer.

[0092]

(3): There is no need, so long as no object types are added, to reconstruct a transform logic library even in a case where methods for the object have changed.

[0093]

A handful of desirable application embodiments of the present invention have been explained above, although the present invention can also be implemented based on various other /11 morphologies conceived by altering, modifying, and/or improving them. It is possible, for example, to construct the client stubs (8) and (53) within the main program (3) in the application embodiments shown in Figures 4, 6, and 7.

Brief explanation of the figures

Figure 1: A block diagram which shows a normal program model for a local computer of the prior art.

Figure 2: A block diagram which shows a normal program model for a local computer which lays the foundation for building the remote object access system provided by the first application embodiment of the present invention.

Figure 3: A flow chart which shows a process whereby the RPC transform logic client stub and RPC server library of the transform logic are generated from the transform logic library shown in Figure 2.

Figure 4: A block diagram which shows the program model of the remote object access system of the first application embodiment.

Figure 5: A block diagram which shows a normal program model for a local computer which lays the foundation for building the remote object access system provided by the second application embodiment of the present invention.

Figure 6: A block diagram which shows the program model of the remote object access system provided by the first [sic: Presumably "second"] application embodiment.

Figure 7: A block diagram which shows the program model of the remote object access system provided by the third application embodiment.

Figure 8: A diagram which shows the constitution of the object of the third application embodiment.

(Explanation of notations)

(2) and (57): Server objects;

(3): Main program;

(4) and (56): Object libraries;

(5): Transform logic library;

(6) and (10): Correspondence tables between pointers and IDs of objects;

(8) and (53): RPC client stubs of the transform logic;

(9) and (54): RPC server libraries of the transform logic;

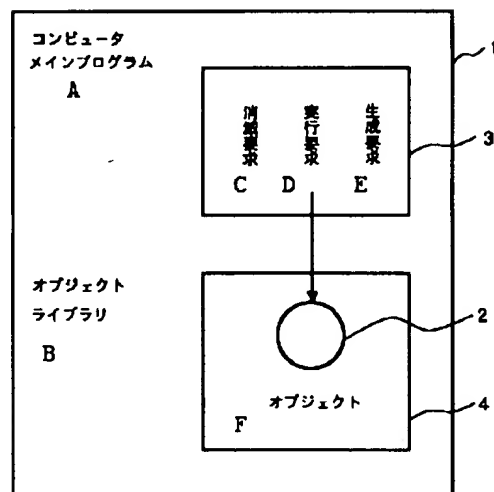
(11), (31), and (41): Server [sic: Presumably "Client"] computers;

(12), (32), and (42): Server computers;

(13): RPC mechanism;

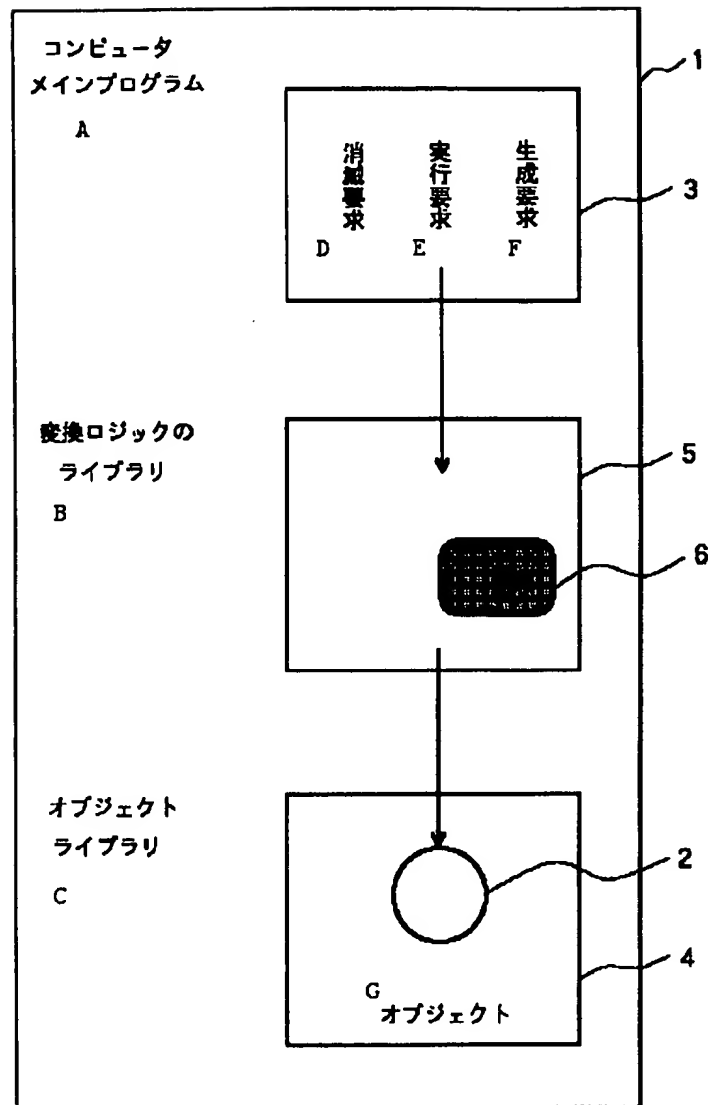
(14) and (51): Client object libraries;
(15) and (52): Client object;
(55): Correspondence table between the type-specific ID and
type of an object.

Figure 1



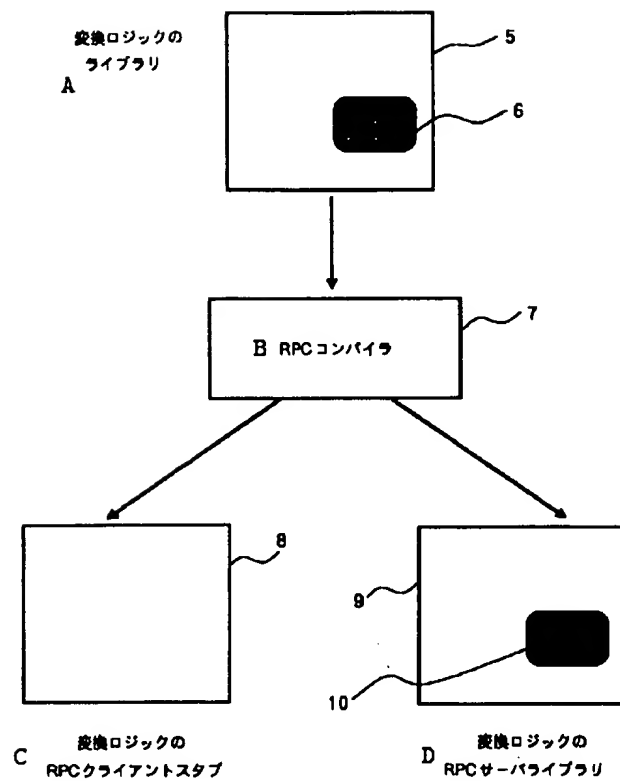
[(A): Computer main program; (B): Object library; (C): Deletion request; (D): Execution request; (E): Generation request; (F): Object]

Figure 2



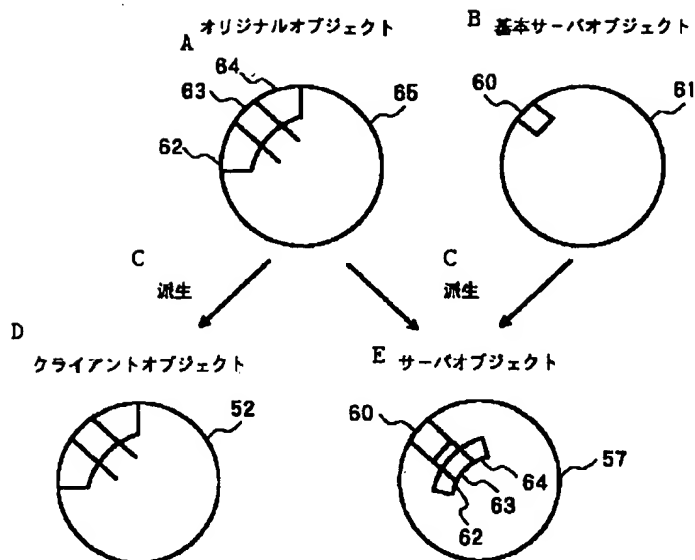
[(A): Computer main program; (B): Library for transform logic;
(C): Object library; (D): Deletion request; (E): Execution
request; (F): Generation request; (G): Object]

Figure 3



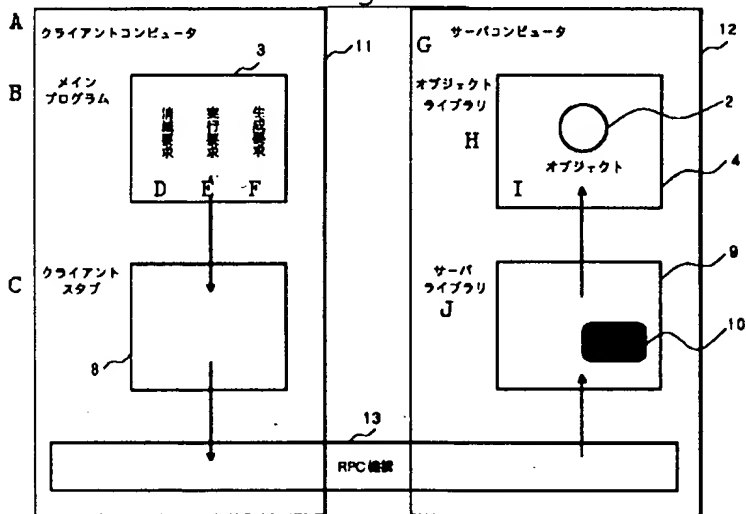
[(A): Library for transform logic; (B): RPC compiler; (C): RPC client stub for transform logic; (D): RPC server library for transform logic]

Figure 8



[(A): Original object; (B): Foundational server object; (C): Branching; (D): Client object; (E): Server object]

Figure 4

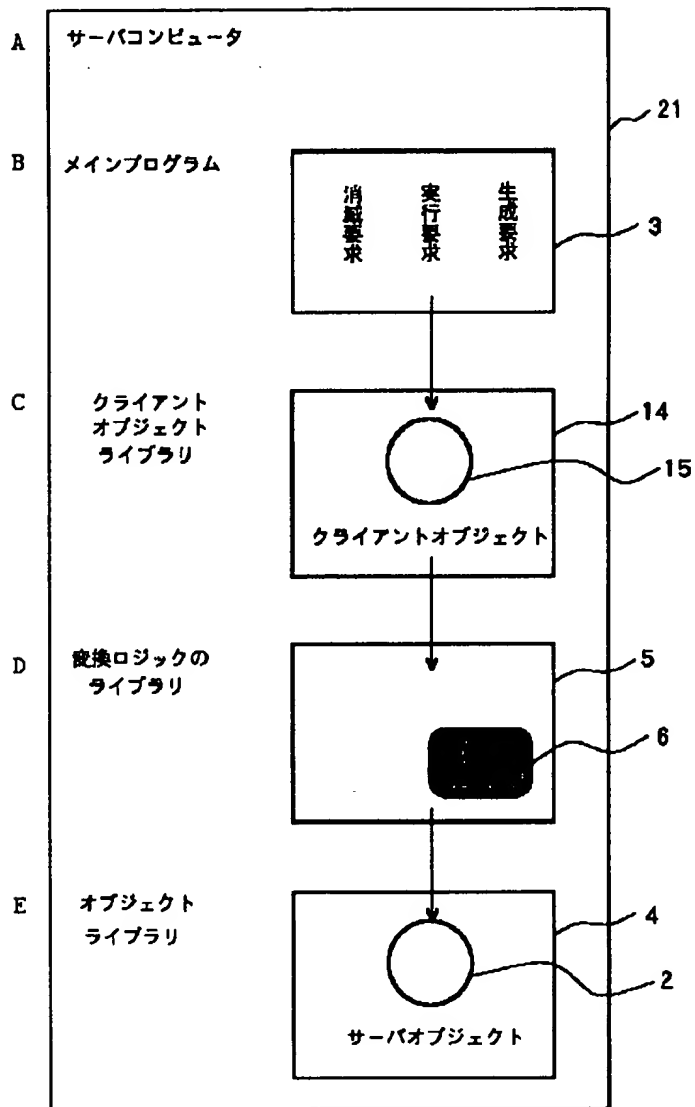


[(A): Client computer; (B): Main program; (C): Client stub; (D): Deletion request; (E): Execution request; (F): Generation request;

(G): Server computer; (H): Object library; (I): Server library;
 (13): RPC mechanism]

Figure 5

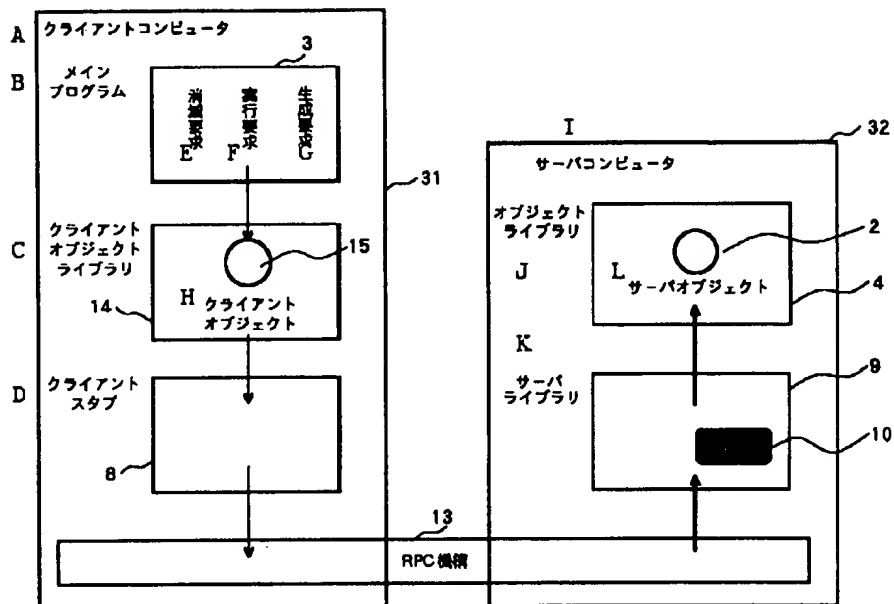
/13



[(A): Server computer; (B): Main program; (C): Client object library; (D): Transform logic library; (E): Object library; (F):

Deletion request; (G): Execution request; (H): Generation request;
 (I): Client object; (J): Server object]

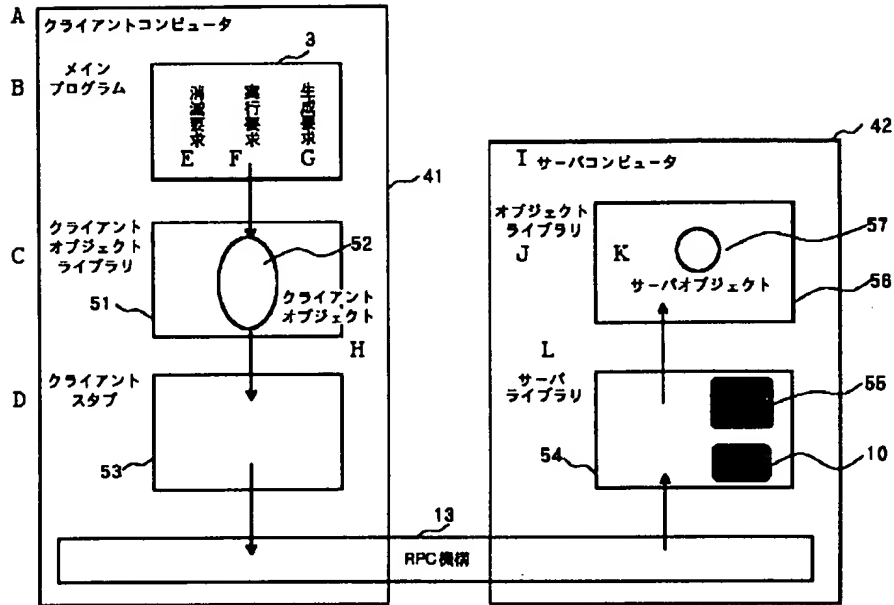
Figure 6



[(A): Client computer; (B): Main program; (C): Client object library; (D): Client stub; (E): Deletion request; (F): Execution request; (G): Generation request; (H): Client object; (I): Server computer; (J): Object library; (K): Server library; (13): RPC mechanism]

Figure 7

/14



[(A): Client computer; (B): Main program; (C): Client object library; (D): Client stub; (E): Deletion request; (F): Execution request; (G): Generation request; (H): Client object; (I): Server computer; (J): Object library; (K): Server library; (13): RPC mechanism]

PAT-NO: JP02001005704A

DOCUMENT-IDENTIFIER: JP 2001005704 A

TITLE: SYSTEM AND METHOD FOR DATABASE ACCESS AND STORAGE MEDIUM

PUBN-DATE: January 12, 2001

INVENTOR-INFORMATION:

NAME	COUNTRY
KODERA, SUNAO	N/A

*xlation Request
Submitted*

ASSIGNEE-INFORMATION:

NAME	COUNTRY
NEC SOFTWARE CHUGOKU LTD	N/A

APPL-NO: JP11172028

APPL-DATE: June 18, 1999

INT-CL (IPC): G06F012/00, G06F017/30

ABSTRACT:

PROBLEM TO BE SOLVED: To increase the efficiency of data access and to simplify a program by generating a data table of unique keys permitting direct access to data that an application requires on a database.

SOLUTION: A database management system 5 controls and accesses the database 4 stored with data. A stored procedure executing means 6 executes a stored procedure registered in the database management system 5 and generates the data table of unique keys defined in the stored procedure. A database access request means 7 requests the database management system 5 to access the database 4 by using the data table of unique keys as the execution result of the stored procedure executing means 6 and receives the request result from the database management system 5.

COPYRIGHT: (C)2001 JPO

(19) 日本国特許庁 (J P)

(12) 公開特許公報 (A)

(11) 特許出願公開番号

特開2001-5704

(P2001-5704A)

(43) 公開日 平成13年1月12日 (2001.1.12)

(51) Int.Cl. ⁷	識別記号	F I	テマート* (参考)
G 0 6 F 12/00 17/30	5 1 3	G 0 6 F 12/00 15/40 15/403	5 1 3 J 5 B 0 7 5 3 1 0 F 5 B 0 8 2 3 8 0 D 3 2 0 Z

審査請求 有 請求項の数 7 O L (全 11 頁)

(21) 出願番号 特願平11-172028

(22) 出願日 平成11年6月18日 (1999.6.18)

(71) 出願人 000211329

中国日本電気ソフトウェア株式会社
広島県広島市南区稲荷町4番1号

(72) 発明者 小寺 直

広島県広島市南区稲荷町4番1号 中国日
本電気ソフトウェア株式会社内

(74) 代理人 100082935

弁理士 京本 直樹 (外2名)

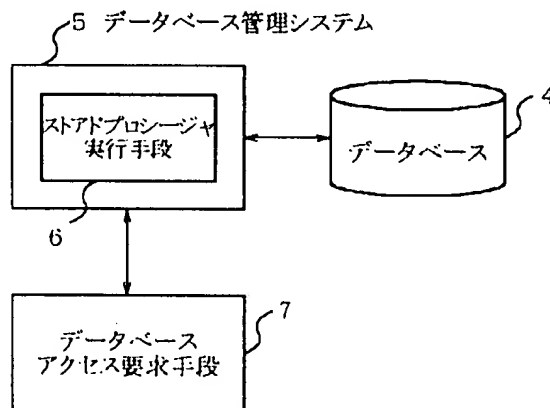
Fターム(参考) 5B075 NK02 NK10 NK24 NK54 PQ02
PQ05
5B082 BA09 EA05 CA08

(54) 【発明の名称】 データベースアクセス方式、方法および記憶媒体

(57) 【要約】

【課題】 データベース上で、アプリケーションが必要とするデータへの直接アクセスが可能な一意キーのデータテーブルを作成することで、データアクセスの効率アップとプログラムの簡略化を実現する。

【解決手段】 データベース管理システム5は、データを記憶蓄積するデータベース4を管理しアクセスする。ストアードプロシージャ実行手段6は、データベース管理システム5に登録されたストアードプロシージャを実行し、ストアードプロシージャに定義された一意キーのデータテーブルを作成する。データベースアクセス要求手段7は、ストアードプロシージャ実行手段6の実行結果である一意キーのデータテーブルを利用してデータベース4にアクセスすることをデータベース管理システム5に要求し要求結果をデータベース管理システム5から受け取る。



【特許請求の範囲】

【請求項1】 データを記憶蓄積するデータベースと、データベースアクセス要求手段からの要求を受け付け前記データベースにアクセスしアクセス結果を前記データベースアクセス要求手段に返すデータベース管理システムと、前記データベース管理システムに含まれストアードプロシージャを実行しストアードプロシージャに定義された一意キーの表を作成するストアードプロシージャ実行手段と、前記ストアードプロシージャ実行手段が作成した一意キーの表を参照して前記データベースにアクセスする要求を前記データベース管理システムに行うデータベースアクセス要求手段と、を備えることを特徴とするデータベースアクセス方式。

【請求項2】 サーバとクライアントがネットワークを介して接続されるクライアントサーバシステムにおいて、前記サーバは、データを記憶蓄積し1つ以上の実表から成る関係データベースと、アプリケーションから要求された処理に基づき前記関係データベースにアクセスし処理結果を前記アプリケーションに返す関係データベース管理システムと、前記実表と関係づけた仮想表を定義し一意キーの表を作成するストアードプロシージャと、前記ストアードプロシージャを起動するストアードプロシージャ起動手段とを備え、前記クライアントは、前記ストアードプロシージャが定義する仮想表を指定して前記関係データベースにアクセスする要求を前記関係データベース管理システムに発行し要求結果を前記関係データベース管理システムから受け取るアプリケーションを備えることを特徴とするデータベースアクセス方式。

【請求項3】 前記ストアードプロシージャ起動手段は、前記ストアードプロシージャが既に起動実行され且つその後前記関係データベースの更新がなされていない場合には前記ストアードプロシージャを起動しないことを特徴とする請求項2記載のデータベースアクセス方式。

【請求項4】 前記ストアードプロシージャが定義する仮想表は、アプリケーションが前記関係データベースにアクセスするときにインデックスの役目をすることを特徴とする請求項2記載のデータベースアクセス方式。

【請求項5】 サーバとクライアントがネットワークを介して接続されるクライアントサーバシステムにおいてデータベースにアクセスする方法であって、アプリケーションは実表とストアードプロシージャで定義されている仮想表とを関係づけた表を定義して関係データベースに対するアクセス要求を関係データベース管理システムに行い、前記関係データベース管理システムは前記アプリケーションからのアクセス要求を分析し、前記ストアードプロシージャが定義生成する仮想表が指定されているときはストアードプロシージャを起動実行してストアードプロシージャに定義された仮想表を生成し、前記仮想表を使用したアプリケーションのアクセス要求を処理してアプリケーションに処理結果を返し、アプリケーションは

関係データベース管理システムから処理結果を受け取ることを特徴とするデータベースアクセス方法。

【請求項6】 前記ストアードプロシージャは基本データの実表とその履歴を保持する2つ以上の実表とから最新の履歴の一意キーから成る仮想表を生成する処理であることを特徴とする請求項5記載のデータベースアクセス方法。

【請求項7】 アプリケーションが実表とストアードプロシージャで定義されている仮想表とを関係づけた表を定義して関係データベースに対するアクセス要求を関係データベース管理システムに行うアクセス要求処理と、前記関係データベース管理システムが前記アプリケーションからのアクセス要求を分析し、前記ストアードプロシージャが定義生成する仮想表が指定されているときは前記ストアードプロシージャを起動するストアードプロシージャ起動処理と、前記ストアードプロシージャを実行してストアードプロシージャに定義された仮想表を生成する仮想表生成処理と、前記仮想表を使用したアプリケーションのアクセス要求を処理してアプリケーションに処理結果を返すアクセス処理と、アプリケーションが前記関係データベース管理システムから処理結果を受け取るデータ受取処理と、をコンピュータに実行させるためのプログラムを記録したことを特徴とする記録媒体。

【発明の詳細な説明】

【0001】

【発明の属する技術分野】本発明は、データベース管理システムを用いてデータベースにアクセスするデータベースアクセス方式、方法および記憶媒体に関し、特にデータベース管理システムに登録したストアードプロシージャを用いてデータベースにアクセスするデータベースアクセス方式、方法および記憶媒体に関する。

【0002】

【従来の技術】データベース管理システムの一つである関係データベース管理システム（以下、RDBMSと略称する）では、関係データベース（以下、RDBと略称する）を表形式で管理している。表には、実際にデータが格納されている実表とデータは保存されず実行時に元になる実表のデータから生成される仮想表（ビューとも称す）がある。アプリケーションは表を参照するSQL言語を用いてRDBMSを介してRDB上の必要なデータを取得する。

【0003】クライアント／サーバシステム（以下、C/Sシステムと略称する）においては、サーバにRDBを接続してRDBMSを備え、クライアント側に存在するアプリケーションがSQL言語を用いてRDBMSに対してRDBアクセス要求を行い、RDBMSからアクセス結果を貰う形態になっている。

【0004】すなわち、クライアントのアプリケーションが必要なデータを検索する場合、アプリケーションはSQL言語の書式にあわせた検索条件式をサーバのRDB

10

20

30

40

50

BMSに発行し、RDBMSがRDBのデータを検索して与えられた検索条件に合致するデータを選択し、クライアントのアプリケーションに検索結果を返す。このとき、RDBMSにおいて、検索用インデックスの作成などの高速アクセス技法が使用される。

【0005】しかし、RDBMSで作成されるインデックスがアプリケーションに最適な形で作成されるとは限らない。

【0006】例えば、図10(a)、(b)に示すような実表Aと実表Bから成るRDBにおいて、商品ごとに最新の値段を求めるアプリケーションについて考えてみる。

【0007】アプリケーションは、以下の処理を行う。

(1) 実表Bにおいて、同一商品番号の中で日付時刻が最新のものを含む行を選択する。

(2) 実表Aと上記(1)の結果を、それぞれの商品番号で関係づける。

(3) 上記(2)の関係づけにより、商品番号と商品名称と値段から成る表を生成する。

(4) 上記(3)の結果である表にアクセスして、商品番号と商品名称と値段のデータを取得する。

【0008】この場合、RDBMSにおいて日付時刻のインデックスは作成されるが、同一商品番号の中で日付時刻が最新のものに対するインデックスは作成されない。このため、アプリケーションは同一商品番号の中で日付時刻が最新のものを選択するための処理手続を、上記(1)において記述しなければならない。このように、RDBMSが作成するインデックスは個々のアプリケーションに最適なものではない。

【0009】また、上記のアプリケーション処理は各クライアントで実行されるので、各クライアントからの要求情報とサーバからの結果情報が、ネットワークを介してサーバとクライアント間で頻繁に行き交うことになる。

【0010】

【発明が解決しようとする課題】上述した従来の技術では、RDBMSで作成されるインデックスはデータ項目の単純なインデックスであり、アプリケーション側で特に必要とするデータがインデックス形式にならずアプリケーションの作成が煩雑になるという問題点がある。また、そのために、RDBMSとアプリケーションとの間のトラフィックが高まるという問題点もある。

【0011】本発明の目的は、RDBMSに登録されたストアドプロシージャを用いて、アプリケーションごとに必要なインデックスを作成し、アプリケーション作成を簡易化し、システム全体の性能を向上させる手段を提供することにある。

【0012】

【課題を解決するための手段】本願第1の発明のデータベースアクセス方式は、データを記憶蓄積するデータ

ベースと、データベースアクセス要求手段からの要求を受け付け前記データベースにアクセスしアクセス結果を前記データベースアクセス要求手段に返すデータベース管理システムと、前記データベース管理システムに含まれストアドプロシージャを実行しストアドプロシージャに定義された一意キーの表を作成するストアドプロシージャ実行手段と、前記ストアドプロシージャ実行手段が作成した一意キーの表を参照して前記データベースにアクセスする要求を前記データベース管理システムに行うデータベースアクセス要求手段と、を備える。

【0013】本願第2の発明のデータベースアクセス方式は、サーバとクライアントがネットワークを介して接続されるクライアントサーバシステムにおいて、前記サーバは、データを記憶蓄積し1つ以上の実表から成る関係データベースと、アプリケーションから要求された処理に基づき前記関係データベースをアクセスし処理結果を前記アプリケーションに返す関係データベース管理システムと、前記実表と関係づけた仮想表を定義し一意キーの表を作成するストアドプロシージャと、前記ストアドプロシージャを起動するストアドプロシージャ起動手段とを備え、前記クライアントは、前記ストアドプロシージャが定義する仮想表を指定して前記関係データベースにアクセスする要求を前記関係データベース管理システムに発行し要求結果を前記関係データベース管理システムから受け取るアプリケーションを備える。

【0014】本願第3の発明のデータベースアクセス方式は、第2の発明において前記ストアドプロシージャ起動手段は、前記ストアドプロシージャが既に起動実行され且つその後に前記関係データベースの更新がなされていない場合には前記ストアドプロシージャを起動しないことを特徴とする。

【0015】本願第4の発明のデータベースアクセス方式は、第2の発明において前記ストアドプロシージャが定義する仮想表は、アプリケーションが前記関係データベースをアクセスするときにインデックスの役目をすることを特徴とする。

【0016】本願第5の発明のデータベースアクセス方法は、サーバとクライアントがネットワークを介して接続されるクライアントサーバシステムにおいてデータベースにアクセスする方法であって、アプリケーションは実表とストアドプロシージャで定義されている仮想表とを関係づけた表を定義して関係データベースに対するアクセス要求を関係データベース管理システムに行い、前記関係データベース管理システムは前記アプリケーションからのアクセス要求を分析し、前記ストアドプロシージャが定義生成する仮想表が指定されているときはストアドプロシージャを起動実行してストアドプロシージャに定義された仮想表を生成し、前記仮想表を使用したアプリケーションのアクセス要求を処理してアプリケーションに処理結果を返し、アプリケーションは関係

データベース管理システムから処理結果を受け取ること
を特徴とする。

【0017】本願第6の発明のデータベースアクセス
方法は、第5の発明において前記ストアプロシージャ
は基本データの実表とその履歴を保持する2つ以上の実
表とから最新の履歴の一意キーから成る仮想表を生成す
る処理であることを特徴とする。

【0018】本願第7の発明の記録媒体は、アプリケー
ションが実表とストアプロシージャで定義されている
仮想表とを関係づけた表を定義して関係データベースに
対するアクセス要求を関係データベース管理システムに
行うアクセス要求処理と、前記関係データベース管理シ
ステムが前記アプリケーションからのアクセス要求を分
析し、前記ストアプロシージャが定義生成する仮想表
が指定されているときは前記ストアプロシージャを起
動するストアプロシージャ起動処理と、前記ストア
プロシージャを実行してストアプロシージャに定義さ
れた仮想表を生成する仮想表生成処理と、前記仮想表を
使用したアプリケーションのアクセス要求を処理してア
プリケーションに処理結果を返すアクセス処理と、ア
プリケーションが前記関係データベース管理システムから
処理結果を受け取るデータ受取処理と、をコンピュータ
に実行させるためのプログラムを記録したことを特徴と
する。

【0019】

【発明の実施の形態】本発明のデータベースアクセス方
式について、図1を参照して説明する。

【0020】図1を参照すると、本発明のデータベース
アクセス方式は、データベース4とデータベース管理シ
ステム5とストアプロシージャ実行手段6とデータベ
ースアクセス要求手段7から構成されている。

【0021】データベース4は、データを記憶蓄積する
データベースである。

【0022】データベース管理システム5は、データベ
ース4を管理し、データベースアクセス要求手段7から
の要求に基づきデータベース4にアクセスしてアクセス
結果をデータベースアクセス要求手段7に返す。

【0023】ストアプロシージャ実行手段6は、デー
タベース管理システム5に登録されたストアプロシ
ージャを実行する。ストアプロシージャは、データベ
ースアクセス要求手段7が使用する一意キーの表（イン
デックス）を作成する処理を含んでいる。

【0024】データベースアクセス要求手段7は、ス
トアプロシージャ実行手段6を実行させ実行結果を利用
してデータベース4にアクセスすることをデータベース
管理システム5に要求し要求結果をデータベース管理シ
ステム5から受け取る。

【0025】本発明の動作について説明する。なお、デ
ータベースアクセス要求手段7が利用するストアプロ
シージャは、あらかじめ公知の方法で、データベース管

理システム5に登録されているものとする。

【0026】まず、データベースアクセス要求手段7は
ストアプロシージャが作成する一意キーの表（インデ
ックス）を指定してデータベース4に対するアクセス要
求をデータベース管理システム5に送る。

【0027】データベース管理システム5はストアプロ
シージャ実行手段6を用いて指定されたストアプロ
シージャを実行してデータベースアクセス要求手段7が
使用する一意キーの表（インデックス）を作成した後に
指定されたアクセス要求処理を行い、処理結果をデー
タベースアクセス要求手段7に返す。

【0028】データベースアクセス要求手段7はデー
タベース管理システム5から処理結果を受け取る。

【0029】このように、データベース管理システム5
に登録されたストアプロシージャにおいてデータベ
ースアクセス要求手段7が使用する一意キーの表（イン
デックス）を作成することにより、データベースアクセス
要求手段7は求めるデータを高速にデータベース4から
取得することができる。

【0030】本発明のデータベースアクセス方式を適用
した本発明の実施の形態について説明する。

【0031】本発明の第1の実施の形態について、図面
を参照して詳細に説明する。第1の実施の形態は、2つ
の実表から成る商品の最新値段を管理しているRDBを
検索して商品の最新値段を取得して処理するシステムに
関するものである。

【0032】図2は、第1の実施の形態の構成を示す図
である。

【0033】図2を参照すると、第1の実施の形態は、
サーバ1とクライアント2がネットワーク3で接続され
ている。サーバ1は、RDB11を接続し、RDBMS
12を備えている。RDBMS12はストアプロシ
ージャ13とストアプロシージャ起動手段14とを含
む。クライアント2は、アプリケーション21を含む。

【0034】RDB11は、データを記憶蓄積する関係
データベースで、実表A111と実表B112とから成
っている。実表A111は基本データであり、実表B1
12は基本データの履歴を保持するものである。

【0035】実表A111は、RDB11を構成してい
る表の一つである。実表A111の構成を図3(a)に
示す。実表A111は商品番号と商品名称とから成り、
各行は主キーである商品番号により一意に区別される。

【0036】実表B112は、RDB11を構成してい
る表の一つである。実表B112の構成を図3(b)に
示す。実表B112は商品番号と日付時刻と値段とから
成り、商品の最新の値段を示している。商品の値段が
変更される度に、変更した日付時刻と新しい値段を示す行
が登録される。商品番号は実表A111の商品番号と対
応しており、これにより実表A111と実表B112が
関係づけられる。日付時刻は値段が登録される日付と時

刻であり、同一商品番号内での値段の登録順番を示すものである。日付時刻に替えて一連番号でもよい。商品番号と日付時刻を連結したもの（以下、「商品番号+日付時刻」と記す）で各行を一意に区別する。

【0037】RDBMS12は、RDB11を管理しアクセスする。アプリケーション21から発行された要求を処理し、発行された要求に対応する処理結果をアプリケーション21に返す。

【0038】ストアプロシージャ13は、実表A111と実表B112を基に仮想表A131を定義生成してデータセットを選択作成する処理を行う。ストアプロシージャ13は、ストアプロシージャ起動手段14により起動される。仮想表A131の名前は、ストアプロシージャ13をRDBMS12に登録するときに決定される。アプリケーション21はこの名前を指定することで、ストアプロシージャ13が生成した仮想表A131を参照することができる。なお、ストアプロシージャ13をRDBMS12に登録する方法については、公知の方法による。

【0039】仮想表A131は、最新の履歴の一意キーから成る仮想表である。実表A111と実表B112を基にストアプロシージャ13が定義生成し、実表A111の商品番号とそれに対応する実表B112の商品番号内で最新（最大）の日付時刻とを組み合わせる選択条件が指定してある。仮想表A131の構成を図3(c)に示す。仮想表A131は商品番号と日付時刻とから成り、商品番号は実表A111の商品番号に、日付時刻は実表B112の同一商品番号内の最新（最大）の日付時刻に、それぞれ対応している。仮想表A131は、アプリケーション21から見れば、常にRDB11の最新状態を反映したインデックスの役目をしており、アプリケーション21から参照される。

【0040】ストアプロシージャ起動手段14は、アプリケーション21がストアプロシージャ13が生成する仮想表A131を使用する要求をRDBMS12に行った場合に、ストアプロシージャ13を起動する。指定されたストアプロシージャ13が既に起動実行されていて且つその後RDB11の更新がなされていない場合には、ストアプロシージャ13を起動しない。この場合、既に実行されたストアプロシージャ13の処理結果が使用される。

【0041】アプリケーション21は、RDB11の最新データを取得するために、ストアプロシージャ13が定義生成する仮想表A131と関係づけを行った仮想表B211を定義してアクセス要求をRDBMS12に発行する。発行した要求に対応する処理結果がRDBMS12から返され、取得したデータを基にデータ処理を行う。

【0042】仮想表B211は、仮想表A131と実表A111と実表B112を基にアプリケーション21が

定義生成する仮想表であり、アプリケーション21が必要とするデータから成る。仮想表B211の構成を図3(d)に示す。仮想表B211は商品番号と商品名称と値段とから成り、商品番号は仮想表A131の商品番号に対応する実表A111の商品番号に、商品名称は仮想表A131の商品番号に対応する実表A111の商品名称に、値段は仮想表A131の「商品番号+日付時刻」に対応する実表B112の値段に、それぞれ対応している。

10 【0043】本発明の第1の実施の形態の動作について、図1～図5を参照して詳細に説明する。

【0044】図4は第1の実施の形態の動作を説明する図で、(a)は実表A111の具体例を、(b)は実表B112の具体例を、(c)は仮想表A131の具体例を、(d)は仮想表B211の具体例を、示したものである。(a)と(b)から、商品名称"あああ"の最新の値段は日付時刻"05261800"に設定された"190"であることがわかる。図5は第1の実施の形態の動作の流れを示す図である。

20 【0045】図5を参照すると、アプリケーション21は、実表A111の商品番号と仮想表A131の商品番号とを関係づけ、実表B112の「商品番号+日付時刻」と仮想表A131の「商品番号+日付時刻」とを関係づけ、商品番号と商品名称と値段とから成る仮想表B211を定義して、RDB11に対するアクセスをRDBMS12に要求する(ステップA11～A12)。

【0046】RDBMS12は、アプリケーション21からのRDB11に対するアクセス要求を分析し(ステップM11)、ストアプロシージャ13が定義生成する仮想表A131が指定されているときは、ストアプロシージャ13の起動実行が必要であるかを判断し、ストアプロシージャ13の起動実行が必要であることを認識したときにストアプロシージャ13を起動する(ステップM12～M14)。指定されたストアプロシージャ13が既に起動実行されていて且つその後RDB11の更新がなされていない場合には、ストアプロシージャ13を起動しない。この場合、既に実行されたストアプロシージャ13の処理結果が使用される。

30 【0047】起動されたストアプロシージャ13は以下の処理を実行する。

(1) 実表B112の商品番号ごとに、同一商品番号内で日付時刻が最新のものを含む行を選択する(ステップS11)。

(2) 実表A111の商品番号と上記(1)の結果の商品番号とを関係づける(ステップS12)。

(3) 実表A111の商品番号と上記(1)の結果の日付時刻から成る仮想表A131のデータセットを作成する(ステップS13)。ストアプロシージャ13の出力結果を図4(c)に示す。

【0048】続いて、RDBMS12は仮想表B211に基づく処理を行う(ステップM15)。RDBMS12の処理結果を図4(d)に示す。

【0049】その後、RDBMS12は処理結果をアプリケーション21に返す(ステップM16)。

【0050】アプリケーション21は、RDBMS12から処理結果を受け取り、アプリケーション処理を行う(ステップA13)。

【0051】このようにして、アプリケーション21は、ストアプロシージャ13の出力結果である仮想表A131をインデックス代わりに使用することにより、求めているRDB11の最新のデータを高速に得ることができる。

【0052】次に、本発明の第2の実施の形態について、図面を参照して詳細に説明する。第2の実施の形態は、3つの実表から成る製品の障害内容を管理しているRDBを検索して製品の最新の障害内容を取得して処理するシステムに関するものである。

【0053】図6は、第2の実施の形態の構成を示す図である。

【0054】図6を参照すると、第2の実施の形態は、サーバ1とクライアント2がネットワーク3で接続され、サーバ1はRDB15を接続しストアプロシージャ17とストアプロシージャ起動手段14を含むRDBMS16を備え、クライアント2はアプリケーション22を備えて構成され、第1の実施の形態のRDB11とRDBMS12とストアプロシージャ13とアプリケーション21とが、RDB15とRDBMS16とストアプロシージャ17とアプリケーション22とに置換されたものである。

【0055】ここでは、第1の実施の形態と異なるRDB15とRDBMS16とストアプロシージャ17とアプリケーション22とについて説明する。

【0056】RDB15は、データを記憶蓄積する関係データベースで、実表A151と実表B152と実表C153とから成っている。実表A151は基本データであり、実表B152と実表C153は基本データの履歴を保持するものである。

【0057】実表A151は、RDB15を構成している表の一つである。実表A151の構成を図7(a)に示す。実表A151は製品番号と製品名称とから成り、各行は主キーである製品番号により一意に区別される。

【0058】実表B152は、RDB15を構成している表の一つである。実表B152の構成を図7(b)に示す。実表B152は製品番号と枝番Aと受付番号と削除とから成る。製品番号は実表A151の製品番号と対応しており、これにより実表B152と実表A151が関係づけられる。枝番Aは同一製品番号内での受付番号の順番を示すもので、一連番号でも日付時刻でもよい。

「製品番号+枝番A」で各行を一意に区別する、受付番

号は製品を通して採番される一連番号であり、製品の障害が受け付けられるごとに受付番号が登録される。障害の内容は実表C153に登録される。同一製品に対して新たな障害受付が有った場合には、枝番Aを上げて新たな受付番号を登録する。削除は、受付番号が一旦受け付けられた後に削除されたか否かを示すもので、受付番号が削除された場合に"DEL"と示される。

【0059】実表C153は、RDB15を構成している表の一つである。実表C153の構成を図7(c)に示す。実表C153は製品番号と枝番Aと枝番Bと障害内容とから成る。「製品番号+枝番A」は実表B152の「製品番号+枝番A」と対応しており、これにより実表C153と実表B152が関係づけられる。枝番Bは同一「製品番号+枝番A」内での障害内容の順番を示すもので、一連番号でも日付時刻でもよい。「製品番号+枝番A+枝番B」で各行を一意に区別する。障害内容には、受付番号に対応した障害内容が登録される。受け付けた受付番号についての障害内容が更新される場合には、枝番Bを上げて更新された障害内容を登録し別のデータとして履歴管理を行う。

【0060】RDBMS16は、RDB15を管理しアクセスする。アプリケーション22から発行された要求を処理し、発行された要求に対応する処理結果をアプリケーション22に返す。

【0061】ストアプロシージャ17は、実表A151と実表B152と実表C153とを基に仮想表A171を定義生成してデータセットを選択作成する処理を行う。ストアプロシージャ17は、ストアプロシージャ起動手段14により起動される。仮想表A171の名前は、ストアプロシージャ17をRDBMS16に登録するときに決定される。アプリケーション22はこの名前を指定することで、ストアプロシージャ17が生成した仮想表A171を参照することができる。なお、ストアプロシージャ17をRDBMS16に登録する方法については、公知の方法による。

【0062】仮想表A171は、最新の履歴の一意キーから成る仮想表である。実表A151と実表B152と実表C153とを基にストアプロシージャ17が定義生成し、実表A151の製品番号、実表B152の同一製品番号内で最大の枝番Aおよび実表C153の同一「製品番号+枝番A」内で最大の枝番Bとを組み合わせた選択条件を指定してある。仮想表A171の構成を図7(c)に示す。仮想表A171は製品番号と枝番Aと枝番Bとから成り、製品番号は実表A151の製品番号に、枝番Aは実表B152の同一製品番号内で最大の枝番Aに、枝番Bは実表C153の同一「製品番号+枝番A」内で最大の枝番Bに、それぞれ対応している。仮想表A171は、アプリケーション22から見れば、常にRDB15の最新状態を反映したインデックスの役目をしており、アプリケーション22から参照される。

11

【0063】アプリケーション22は、RDB15の最新データを取得するために、ストアプロシージャ17が定義生成する仮想表A171と関係づけを行った仮想表B221を定義してアクセス要求をRDBMS16に発行する。発行した要求に対応する処理結果がRDBMS16から返され、取得したデータを基にデータ処理を行う。

【0064】仮想表B221は、仮想表A171と実表A151と実表B152と実表C153を基にアプリケーション22が定義生成する仮想表であり、アプリケーション22が必要とするデータから成る。仮想表B221の構成を図7(d)に示す。仮想表B221は製品番号と製品名称と受付番号と障害内容とから成り、製品番号は仮想表A171の製品番号に対応する実表A151の製品番号に、製品名称は仮想表A171の製品番号に対応する実表A151の製品名称に、受付番号は仮想表A171の「製品番号+枝番A」に対応する実表B152の受付番号に、障害内容は仮想表A171の「製品番号+枝番A+枝番B」に対応する実表C153の障害内容に、それぞれ対応している。

【0065】本発明の第2の実施の形態の動作について、図6～図9を参照して詳細に説明する。

【0066】図8は第2の実施の形態の動作を説明する図で、(a)は実表A151の具体例を、(b)は実表B152の具体例を、(c)は実表C153の具体例を、(d)は仮想表A171の具体例を、(e)は仮想表B221の具体例を、示したものである。(b)において、受付番号“002”および“003”は、一旦受け付けられたがその後削除されたので、削除に“DEL”と示されている。図9は第2の実施の形態の動作の流れを示す図である。

【0067】図9を参照すると、アプリケーション22は、実表A151の製品番号と仮想表A171の製品番号とを関係づけ、実表B152の「製品番号+枝番A」と仮想表A171の「製品番号+枝番A」とを関係づけ、実表C153の「製品番号+枝番A+枝番B」と仮想表A171の「製品番号+枝番A+枝番B」とを関係づけ、製品番号と製品名称と受付番号と障害内容とから成る仮想表B221を定義して、RDB15に対するアクセスをRDBMS16に要求する(ステップA21～A22)。

【0068】RDBMS16は、アプリケーション22からのRDB15に対するアクセス要求を分析し(ステップM21)、ストアプロシージャ17が定義生成する仮想表A171が指定されているときは、ストアプロシージャ17の起動実行が必要であるか否かを判断し、ストアプロシージャ17の起動実行が必要であることを認識したときにストアプロシージャ17を起動する(ステップM22～M24)。指定されたストアプロシージャ17が既に起動実行されていて且つその後

12

にRDB15の更新がなされていない場合には、ストアプロシージャ17を起動しない。この場合、既に実行されたストアプロシージャ17の処理結果が使用される。

【0069】起動されたストアプロシージャ17は以下の処理を実行する。

(1) 実表C153の「製品番号+枝番A」ごとに、同一「製品番号+枝番A」内で枝番Bが最大のものを含む行を選択する(ステップS21)。この例では、図8

(c)の右端に○印が付いている行が選択される。

(2) 実表B152の「製品番号+枝番A」と上記(1)の結果の「製品番号+枝番A」とを関係づける(ステップS22)。このとき、実表B152の削除に“DEL”が設定されている実表B152の「製品番号+枝番A」は除く。この例では、受付番号“002”および“003”の行は削除に“DEL”が設定されているので除かれ、関係づけられるのは、図8(b)および(c)の右端に◎印が付いている行である。

(3) 実表A151の製品番号と上記(2)の結果の枝番Aと上記(2)の結果の枝番Bとから成る仮想表A171のデータセットを作成する(ステップS23)。ストアプロシージャ17の出力結果を図8(d)に示す。

【0070】続いて、RDBMS16は仮想表B221に基づく処理を行う(ステップM25)。RDBMS16の処理結果を図8(e)に示す。

【0071】その後、RDBMS16は処理結果をアプリケーション22に返す(ステップM26)。

【0072】アプリケーション22は、RDBMS16から処理結果を受け取り、アプリケーション処理を行う(ステップA23)。

【0073】このようにして、アプリケーション22は、ストアプロシージャ17の出力結果である仮想表A171をインデックス代わりに使用することにより、求めているRDB15の最新のデータを高速に得ることができる。すなわち、アプリケーション22は複雑な処理をすることなく、製品番号に対する最新の受付番号と障害内容とを取得して、処理を行うことができる。

【0074】本発明による上述した実施の形態において、データベースアクセス方式の処理動作を実行するためのプログラム等を、データとして磁気ディスクや光ディスク等の記憶装置(図示せず)に記憶するようにし、記憶されたデータを読み出してデータベースアクセス方式を動作させるために用いる。このように、本発明によるデータベースアクセス方式を動作させるデータを記憶媒体に記憶させ、この記憶媒体をインストールすることによりデータベースアクセス方式の機能が実現できるようになる。

【0075】

【発明の効果】第1の効果は、必要なストアプロシ

10

20

30

40

50

13

ジヤを参照するだけで、アプリケーションから常に最新のデータを高速に参照することができることである。その理由は、アプリケーション用のインデックスを生成する機能をストアードプロシージャの中に構成するような手段を設けたためである。

【0076】第2の効果は、クライアントの負荷が軽減され且つネットワークのトラフィックも減少することである。その理由は、ストアードプロシージャがサーバー側で実行されるので、アプリケーションから複雑なSQLを発行する必要がなく、アプリケーションも一意キーの表を基にしたデータ操作処理を記述するだけで処理が構築でき、クライアントのアプリケーションとサーバーのRDBMSとの間に発生するやりとりも少なくて済むためである。

【0077】第3の効果は、データベースの障害時、アプリケーション用のインデックスのために物理的なインデックステーブルの再作成といった特別の復旧処置が要らないことである。その理由は、アプリケーション用のインデックスが仮想的なテーブルとしてストアードプロシージャを利用しているためである。

【図面の簡単な説明】

【図1】本発明を説明する図

【図2】第1の実施の形態の構成を示す図

【図3】第1の実施の形態における(a)実表A(b)実表B(c)仮想表A(d)仮想表Bの構成を示す図

【図4】第1の実施の形態の動作を説明する(a)実表A(b)実表B(c)仮想表A(d)仮想表Bの図

【図5】第1の実施の形態の動作の流れを示す図

【図6】第2の実施の形態の構成を示す図

【図7】第2の実施の形態における(a)実表A(b)実表B(c)実表C(d)仮想表A(e)仮想表Bの構成を示す図

14

【図8】第2の実施の形態の動作を説明する(a)実表A(b)実表B(c)実表C(d)仮想表A(e)仮想表Bの図

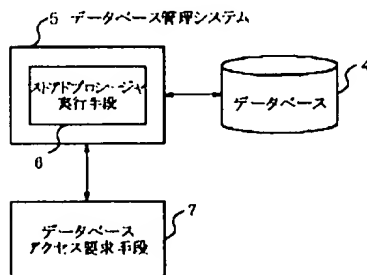
【図9】第2の実施の形態の動作の流れを示す図

【図10】従来の動作を説明する(a)実表A(b)実表B(c)結果の図

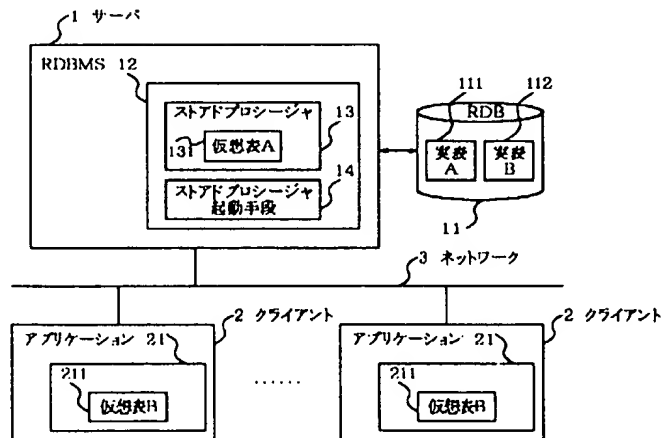
【符号の説明】

- | | |
|-----|-----------------|
| 1 | サーバ |
| 2 | クライアント |
| 3 | ネットワーク |
| 4 | データベース |
| 5 | データベース管理システム |
| 6 | ストアードプロシージャ実行手段 |
| 7 | データベースアクセス要求手段 |
| 11 | RDB |
| 12 | RDBMS |
| 13 | ストアードプロシージャ |
| 14 | ストアードプロシージャ起動手段 |
| 15 | RDB |
| 16 | RDBMS |
| 17 | ストアードプロシージャ |
| 21 | アプリケーション |
| 22 | アプリケーション |
| 111 | 実表A |
| 112 | 実表B |
| 131 | 仮想表A |
| 151 | 実表A |
| 152 | 実表B |
| 153 | 実表C |
| 171 | 仮想表A |
| 211 | 仮想表B |
| 221 | 仮想表B |

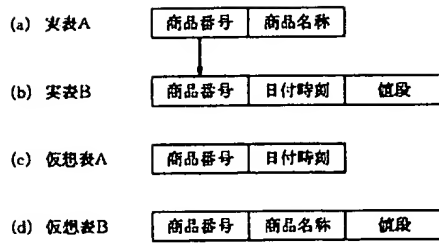
【図1】



【図2】



【図3】



【図4】

(a) 実表A

商品番号	商品名称
1	あああ
2	いゐい
3	ううう
4	えええ
5	おおお

(b) 実表B

商品番号	日付時刻	値段
1	05261000	200
1	05261200	198
1	05261400	195
1	05261600	193
1	05261800	190
2	05261000	100
2	05261800	95
3	05261000	498
4	05261000	300
4	05261200	295
4	05261400	290
5	05261000	98
5	05261800	70

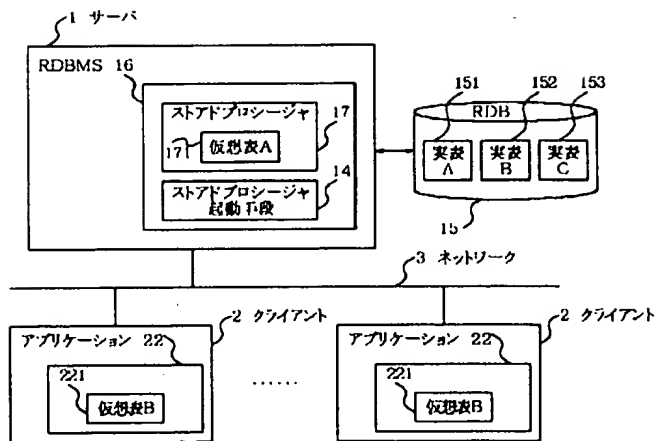
(c) 仮想表A

商品番号	日付時刻
1	05261800
2	05261800
3	05261000
4	05261400
5	05261800

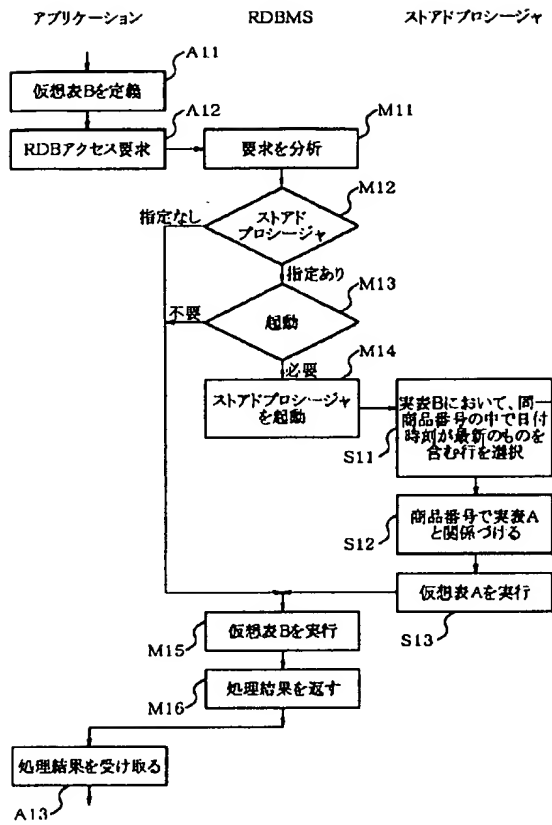
(d) 仮想表B

商品番号	商品名称	値段
1	あああ	190
2	いゐい	85
3	ううう	498
4	えええ	290
5	おおお	70

【図6】



【図5】



【図7】

(a) 実表A

製品番号	製品名称
------	------

(b) 実表B

製品番号	枝番A	受付番号	削除
------	-----	------	----

(c) 実表C

製品番号	枝番A	枝番B	障害内容
------	-----	-----	------

(d) 仮想表A

製品番号	枝番A	枝番B
------	-----	-----

(e) 仮想表B

製品番号	製品名称	受付番号	障害内容
------	------	------	------

【図8】

製品番号	製品名称
1	あいう
2	かきく
3	さしす

(a) 実表A

製品番号	枝番A	受付番号	削除
1	1	001	
2	1	002	DEL
2	2	004	
3	1	003	DEL
3	2	005	

(b) 実表B

製品番号	枝番A	枝番B	障害内容
1	1	1	A
1	1	2	B
2	1	1	C
2	2	1	D
3	1	1	E
3	2	1	F
3	2	2	G
3	2	3	H

(c) 実表C

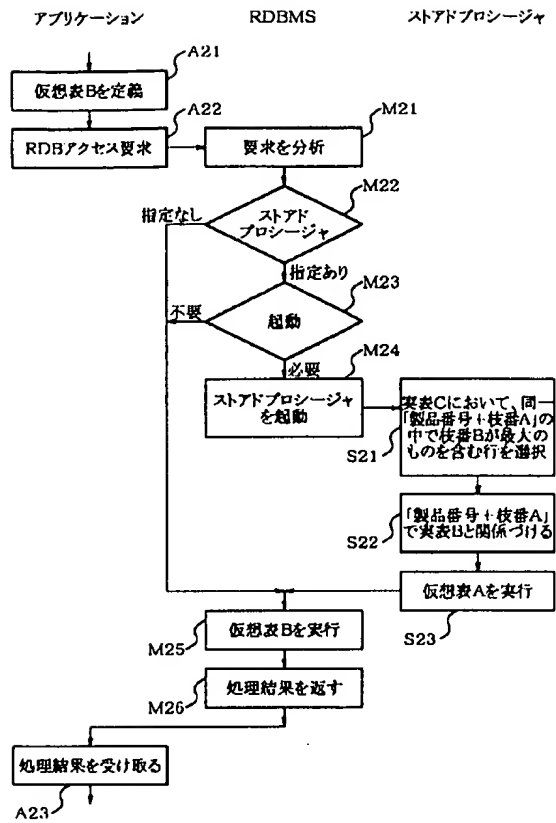
製品番号	枝番A	枝番B
1	1	2
2	2	1
3	2	3

(d) 仮想表A

製品番号	製品名称	受付番号	障害内容
1	あいう	001	B
2	かきく	004	D
3	さしす	005	H

(e) 仮想表B

【図9】



【図10】

商品番号	商品名称
1	あああ
2	いはい
3	ううう
4	えええ
5	おとお

(a) 実表A

商品番号	日付時刻	値段
1	05261000	200
1	05261200	198
1	05261400	195
1	05261600	193
1	05261800	190
2	05261000	100
2	05261600	95
3	05261000	498
4	05261000	300
4	05261200	295
4	05261400	290
5	05261000	98
5	05261800	70

(b) 実表B

商品番号	商品名称	値段
1	あああ	190
2	いはい	95
3	ううう	498
4	えええ	290
5	おとお	70

(c) 結果

JP 2001 005 704 A

Pub. 12 Jan 2001

APPN JP 11172028

* NOTICES *

Japan Patent Office is not responsible for any damages caused by the use of this translation.

1. This document has been translated by computer. So the translation may not reflect the original precisely.
2. **** shows the word which can not be translated.
3. In the drawings, any words are not translated.

DETAILED DESCRIPTION

[Detailed Description of the Invention]

[0001]

[The technical field to which invention belongs] This invention relates to the data base access method, method, and storage which access a data base using the stored procedure registered especially into the data base management system about the data base access method, method, and storage which access a data base using a data base management system.

[0002]

[Description of the Prior Art] With the relational database management system (it is hereafter called RDBMS for short) which is one of the data base management systems, the relational database (it is hereafter called RDB for short) is managed by the tabular format. The real table and data with which data is actually stored have in a table the virtual table (it is also called a view) generated from the data of the real table which is not saved but becomes origin at the time of activation. Application acquires the required data on RDB through RDBMS using the SQL language which refers to a table.

[0003] In the client/server system (it is hereafter called a C/S system for short), RDB is connected to a server and it has RDBMS, and the application which exists in a client side performs a RDB access request to RDBMS using SQL language, and has become the gestalt which gets an access result from RDBMS.

[0004] That is, when searching the data which needs the application of a client, application publishes retrieval conditional expression united with the format of SQL language to RDBMS of a server, chooses the data corresponding to the retrieval conditions to which RDBMS searched the data of RDB and was given, and returns a retrieval result to the application of a client. At this time, rapid access techniques, such as creation of the index for retrieval, are used in RDBMS.

[0005] However, the index created by RDBMS is not necessarily created in the optimal form for application.

[0006] For example, in RDB which consists of the real table A as shown in drawing 10 (a) and (b), and the real table B, the application which asks for the newest price for every goods is considered.

[0007] Application performs the following processings.

- (1) Choose the line in which the date time of day contains the newest thing in the same quotient lot number number in the real table B.
- (2) Connect the result of the real table A and the above (1) in each quotient lot number number.
- (3) Relating of the above (2) generates a quotient lot number number, a goods name, and the table that consists of a price.
- (4) Access the table which it is as a result of the above (3), and acquire the data of a quotient lot number number, a goods name, and a price.

[0008] In this case, although the index of the date time of day is created in RDBMS, the index to the thing of the newest [time of day / date] is not created in the same quotient lot number number. For this reason, application must describe processing procedure for the date time of day to choose the

newest thing in the above (1) in the same quotient lot number number. Thus, the index which RDBMS creates is not the the best for each application.

[0009] Moreover, since the above-mentioned application process is performed by each client, information will go back and forth frequently between a server and a client through a network the result from the demand information and the server from each client.

[0010]

[Problem(s) to be Solved by the Invention] In the Prior art mentioned above, the index created by RDBMS is a simple index of a data item, and has the trouble that the data needed especially by the application side does not become index format, but creation of application becomes complicated. moreover -- therefore, there is also a trouble that the traffic between RDBMS and application increases.

[0011] Using the stored procedure registered into RDBMS, the purpose of this invention creates a required index for every application, simplifies application creation, and is to offer the means which raises the system-wide engine performance.

[0012]

[Means for Solving the Problem] A data base access method of invention of this application 1st A data base which carries out storage are recording of the data, and a database management system which receives a demand from a data base access request means, accesses said data base, and returns an access result to said data base access request means, A stored procedure activation means to create a table of a meaning key which was contained in said database management system, performed a stored procedure, and was defined as a stored procedure, It has a data base access request means to give a demand which accesses said data base with reference to a table of a meaning key which said stored procedure activation means created to said database management system.

[0013] A data base access method of invention of this application 2nd In a client/server system by which a client is connected with a server through a network said server A relational database which carries out storage are recording of the data, and consists of one or more real tables, and a rational database management system which accesses said relational database based on processing demanded from application, and returns a processing result to said application, A stored procedure which defines a virtual table connected with said real table, and creates a table of a meaning key, It has a stored procedure starting means to start said stored procedure. Said client It has application which publishes a demand which specifies a virtual table which said stored procedure defines, and accesses said relational database to said rational database management system, and receives a demand result from said rational database management system.

[0014] A data base access method of invention of this application 3rd is characterized by not starting said stored procedure, when, as for said stored procedure starting means, renewal of said relational database is not made after that by already carrying out starting activation of said stored procedure in the 2nd invention.

[0015] A virtual table where said stored procedure defines a data base access method of invention of this application 4th in the 2nd invention is characterized by carrying out a duty of an index, when application accesses said relational database.

[0016] A data base access method of invention of this application 5th It is the method of accessing a data base in a client/server system by which a client is connected with a server through a network. Application defines a table which connected a virtual table defined by a real table and stored procedure, and performs an access request to a relational database to a rational database management system. Said rational database management system analyzes an access request from said application. A virtual table which carried out starting activation of the stored procedure, and was defined as a stored procedure when a virtual table said stored procedure carries out [a virtual table] definition generation was specified is generated. An access request of application which used said virtual table is processed, a processing result is returned to application, and application is characterized by receiving a processing result from a rational database management system.

[0017] A data base access method of invention of this application 6th is characterized by said stored procedure being processing which generates a virtual table which consists of a meaning key of hysteresis of a real table of the master data, and two real tables or more holding the hysteresis to the newest in the 5th invention.

[0018] Access request processing which a record medium of invention of this application 7th defines a table with which application connected a virtual table defined by a real table and stored procedure, and performs an access request to a relational database to a rational database management system, Said relational database management system analyzes an access request from said application. Stored procedure starting processing in which said stored procedure is started when a virtual table said stored procedure carries out [a virtual table] definition generation is specified, Virtual table generation processing which generates a virtual table which performed said stored procedure and was defined as a stored procedure, Access processing which processes an access request of application which used said virtual table, and returns a processing result to application, It is characterized by recording a program for making a computer perform data receipt processing in which application receives a processing result from said rational database management system.

[0019]

[Embodiment of the Invention] The data base access method of this invention is explained with reference to drawing 1 .

[0020] Reference of drawing 1 constitutes the data base access method of this invention from the data base 4, a database management system 5, a stored procedure activation means 6, and a data base access request means 7.

[0021] A data base 4 is a data base which carries out storage are recording of the data.

[0022] A database management system 5 manages a data base 4, accesses a data base 4 based on the demand from the data base access request means 7, and returns an access result to the data base access request means 7.

[0023] The stored procedure activation means 6 performs the stored procedure registered into the database management system 5. The stored procedure includes the processing which creates the table (index) of the meaning key which the data base access request means 7 uses.

[0024] The data base access request means 7 requires a database management system 5 to perform the stored procedure activation means 6 and to access a data base 4 using an activation result, and receives a demand result from a database management system 5.

[0025] Actuation of this invention is explained. In addition, beforehand, the stored procedure which the data base access request means 7 uses shall be a well-known method, and shall be registered into the database management system 5.

[0026] First, the data base access request means 7 specifies the table (index) of the meaning key which a stored procedure creates, and sends the access request to a data base 4 to a database management system 5.

[0027] A database management system 5 performs access request processing specified after creating the table (index) of the meaning key which performs the stored procedure specified using the stored procedure activation means 6, and the data base access request means 7 uses, and returns a processing result to the data base access request means 7.

[0028] The data base access request means 7 receives a processing result from a database management system 5.

[0029] Thus, the data base access request means 7 can acquire the data for which it asks from a data base 4 at a high speed by creating the table (index) of the meaning key which the data base access request means 7 uses in the stored procedure registered into the database management system 5.

[0030] The gestalt of operation of this invention which applied the data base access method of this invention is explained.

[0031] The gestalt of operation of the 1st of this invention is explained to details with reference to a drawing. The gestalt of the 1st operation searches RDB which has managed the newest price of the

goods which consist of two real tables, and is related with the system which acquires and processes the newest price of goods.

[0032] Drawing 2 is drawing showing the 1st configuration of the gestalt of operation.

[0033] As for the gestalt of the 1st operation, reference of drawing 2 connects the client 2 with the server 1 in the network 3. The server 1 connected RDB11 and is equipped with RDBMS12. RDBMS12 includes a stored procedure 13 and the stored procedure starting means 14. A client 2 contains application 21.

[0034] RDB11 is the relational database which carries out storage and recording of the data, and consists of the real table A111 and the real table B112. The real table A111 is the master data, and the real table B112 holds the hysteresis of the master data.

[0035] The real table A111 is one of the tables which constitute RDB11. The configuration of the real table A111 is shown in drawing 3 (a). The real table A111 consists of a quotient lot number number and a goods name, and each line is distinguished by the meaning by the quotient lot number number which is a major key.

[0036] The real table B112 is one of the tables which constitute RDB11. The configuration of the real table B112 is shown in drawing 3 (b). The real table B112 consists of a quotient lot number number, the date time of day, and a price, and shows the newest price of goods. The line which shows the date time of day changed whenever the price of goods was changed, and a new price is registered. The quotient lot number number corresponds with the quotient lot number number of the real table A111, and, thereby, the real table A111 and the real table B112 are connected. The date time of day is the date and time of day when a price is registered, and shows the registration sequence of the price within the same quotient lot number number. It may change at the date time of day, and the sequence number may be used. Each line is uniquely distinguished by what connected the date time of day with the quotient lot number number (it is hereafter described as "quotient lot number number + date time of day").

[0037] RDBMS12 manages and accesses RDB11. The demand published from application 21 is processed and the processing result corresponding to the published demand is returned to application 21.

[0038] A stored procedure 13 performs processing which definition-generation-carries out selection creation of the data set for a virtual table A131 based on the real table A111 and the real table B112. A stored procedure 13 is started by the stored procedure starting means 14. The identifier of a virtual table A131 is determined when registering a stored procedure 13 into RDBMS12. Refer to the virtual table A131 which the stored procedure 13 generated for application 21 by specifying this identifier. In addition, about the method of registering a stored procedure 13 into RDBMS12, it is based on a well-known method.

[0039] A virtual table A131 is a virtual table which consists of the meaning key of the newest hysteresis. The selection condition with which a stored procedure 13 combines the newest (max) date time of day in the quotient lot number number of the real table B112 corresponding to [definition generation] the quotient lot number number of the real table A111 and it based on the real table A111 and the real table B112 is specified. The configuration of a virtual table A131 is shown in drawing 3 (c). A virtual table A131 consists of a quotient lot number number and the date time of day, and that of the real table A111 supports [the quotient lot number number] the quotient lot number number, respectively at the date time of day of the newest [time of day / date] in the same quotient lot number number of the real table B112 (max). If it sees from application 21, the virtual table A131 will carry out the duty of the index which always reflected the newest condition of RDB11, and will be referred to from application 21.

[0040] The stored procedure starting means 14 starts a stored procedure 13, when the demand whose application 21 uses the virtual table A131 which a stored procedure 13 generates is given to RDBMS12. When starting activation of the specified stored procedure 13 has already been carried out and renewal of RDB11 is not made after that, a stored procedure 13 is not started. In this case, the

processing result of the already performed stored procedure 13 is used.

[0041] In order to acquire the newest data of RDB11, application 21 defines the virtual table A131 as for which a stored procedure 13 carries out definition generation, and the connected virtual table B211, and publishes an access request to RDBMS12. The processing result corresponding to the published demand is returned from RDBMS12, and data processing is performed based on the acquired data.

[0042] Application 21 is a definition generation virtual table based on a virtual table A131, the real table A111, and the real table B112, and a virtual table B211 consists of the data which application 21 needs. The configuration of a virtual table B211 is shown in drawing 3 (d). A virtual table B211 consists of a quotient lot number number, a goods name, and a price, and a quotient lot number number in the quotient lot number number of the real table A111 corresponding to the quotient lot number number of a virtual table A131. The goods name supports the price of the real table B112 corresponding to the "quotient lot number number + date time of day" of a virtual table A131 in a price at the goods name of the real table A111 corresponding to the quotient lot number number of a virtual table A131, respectively.

[0043] Actuation of the gestalt of operation of the 1st of this invention is explained to details with reference to drawing 1 - drawing 5.

[0044] drawing where drawing 4 explains actuation of the gestalt of the 1st operation -- it is -- (a) -- the example of the real table A111 -- in (b), (c) shows the example of a virtual table A131, and (d) shows the example of a virtual table B211 for the example of the real table B112. the goods name from (a) and (b) -- " -- such -- the newest price of ***" -- the date time of day -- it turns out that it is "190" set as "05261800." Drawing 5 is drawing showing the flow of actuation of the gestalt of the 1st operation.

[0045] When drawing 5 is referred to, application 21 The quotient lot number number of the real table A111 and the quotient lot number number of a virtual table A131 are connected. The "quotient lot number number + date time of day" of the real table B112 and the "quotient lot number number + date time of day" of a virtual table A131 are connected, the virtual table B211 which consists of a quotient lot number number, a goods name, and a price is defined, and access to RDB11 is required of RDBMS12 (steps A11-A12).

[0046] It judges whether RDBMS12 needs starting activation of a stored procedure 13, when the virtual table A131 analyzes the access request to RDB11 from application 21 (step M11), and a stored procedure 13 carries out [the virtual table] definition generation is specified, and when it has recognized that a stored procedure 13 needs to be starting performed, a stored procedure 13 is started (steps M12-M14). When starting activation of the specified stored procedure 13 has already been carried out and renewal of RDB11 is not made after that, a stored procedure 13 is not started. In this case, the processing result of the already performed stored procedure 13 is used.

[0047] The started stored procedure 13 performs the following processings.

(1) The date time of day chooses the line containing the newest thing within the same quotient lot number number for every quotient lot number number of the real table B112 (step S11).

(2) Connect the quotient lot number number of the real table A111, and the quotient lot number number as a result of the above (1) (step S12).

(3) Create the quotient lot number number of the real table A111, and the data set of the virtual table A131 which consists of the date time of day as a result of the above (1) (step S13). The output of a stored procedure 13 is shown in drawing 4 (c).

[0048] Then, RDBMS12 performs processing based on a virtual table B211 (step M15). The processing result of RDBMS12 is shown in drawing 4 (d).

[0049] Then, RDBMS12 returns a processing result to application 21 (step M16).

[0050] Application 21 performs reception and an application process for a processing result from RDBMS12 (step A13).

[0051] Thus, application 21 can obtain the newest data of RDB11 for which it is asking at a high speed by using the virtual table A131 which is the output of a stored procedure 13 instead of an index.

[0052] next, gestalt of operation of the 2nd of this invention ***** -- it explains to details with

reference to a drawing. The gestalt of the 2nd operation searches RDB which has managed the contents of a failure of the product which consists of three real tables, and is related with the system which acquires and processes the newest contents of a failure of a product.

[0053] Drawing 6 is drawing showing the 2nd configuration of the gestalt of operation.

[0054] If drawing 6 is referred to, as for the gestalt of the 2nd operation, a client 2 will be connected with a server 1 in a network 3. A server 1 is equipped with RDBMS16 which connects RDB15 and includes a stored procedure 17 and the stored procedure starting means 14. A client 2 is equipped with application 22 and constituted. RDB11, RDBMS12, the stored procedure 13, and application 21 of a gestalt of the 1st operation It is replaced by RDB15, RDBMS16, the stored procedure 17, and application 22.

[0055] Here, RDB15, RDBMS16, the different stored procedure 17, and different application 22 from the gestalt of the 1st operation are explained.

[0056] RDB15 is the relational database which carries out storage are recording of the data, and consists of the real table A151, the real table B152, and the real table C153. The real table A151 is the master data, and the real table B152 and the real table C153 hold the hysteresis of the master data.

[0057] The real table A151 is one of the tables which constitute RDB15. The configuration of the real table A151 is shown in drawing 7 (a). The real table A151 consists of a part number and a product name, and each line is distinguished by the meaning by the part number which is a major key.

[0058] The real table B152 is one of the tables which constitute RDB15. The configuration of the real table B152 is shown in drawing 7 (b). The real table B152 consists of a part number, a branch number A and a receipt number, and deletion. The part number corresponds with the part number of the real table A151, and, thereby, the real table B152 and the real table A151 are connected. A branch number A may show the sequence of the receipt number within the same part number, and the sequence number or the date time of day is sufficient as it. Each line is uniquely distinguished with "the part number + branch number A." A receipt number is the sequence number ****(ed) through a product, and a receipt number is registered whenever the failure of a product is received. The contents of the failure are registered into the real table C153. When there is new failure reception to the same product, a branch number A is raised and a new receipt number is registered. Deletion shows whether it was deleted once the receipt number was received, and when a receipt number is deleted, it is indicated to be "DEL."

[0059] The real table C153 is one of the tables which constitute RDB15. The configuration of the real table C153 is shown in drawing 7 (c). The real table C153 consists of a part number, and a branch number A, a branch number B and the contents of a failure. "The part number + branch number A" corresponds with the "part number + branch number A" of the real table B152, and, thereby, the real table C153 and the real table B152 are connected. A branch number B may show the sequence of the contents of a failure within the same "part number + branch number A", and the sequence number or the date time of day is sufficient as it. Each line is uniquely distinguished with "the part number + branch number A+ branch number B." The contents of a failure corresponding to a receipt number are registered into the contents of a failure. When the contents of a failure about the received receipt number are updated, the contents of a failure which raised the branch number B and were updated are registered, and hysteresis management is performed as another data.

[0060] RDBMS16 manages and accesses RDB15. The demand published from application 22 is processed and the processing result corresponding to the published demand is returned to application 22.

[0061] A stored procedure 17 performs processing which definition-generation-carries out selection creation of the data set for a virtual table A171 based on the real table A151, the real table B152, and the real table C153. A stored procedure 17 is started by the stored procedure starting means 14. The identifier of a virtual table A171 is determined when registering a stored procedure 17 into RDBMS16. Refer to the virtual table A171 which the stored procedure 17 generated for application 22 by specifying this identifier. In addition, about the method of registering a stored procedure 17 into

RDBMS16, it is based on a well-known method.

[0062] A virtual table A171 is a virtual table which consists of the meaning key of the newest hysteresis. The selection condition with which a stored procedure 17 definition-generation-combines the greatest branch number B within the greatest branch number A in the part number of the real table A151 and the same part number of the real table B152 and the same "part number + branch number A" of the real table C153 based on the real table A151, the real table B152, and the real table C153 is specified. The configuration of a virtual table A171 is shown in drawing 7 (c). A virtual table A171 consists of a part number, a branch number A, and a branch number B, a branch number A is equivalent to the greatest branch number A in the same part number of the real table B152, and the branch number B is equivalent to the greatest branch number B at the part number of the real table A151 for the part number, respectively within the same "part number + branch number A" of the real table C153. If it sees from application 22, the virtual table A171 will carry out the duty of the index which always reflected the newest condition of RDB15, and will be referred to from application 22.

[0063] In order to acquire the newest data of RDB15, application 22 defines the virtual table A171 as for which a stored procedure 17 carries out definition generation, and the connected virtual table B221, and publishes an access request to RDBMS16. The processing result corresponding to the published demand is returned from RDBMS16, and data processing is performed based on the acquired data.

[0064] Application 22 is a definition generation virtual table based on a virtual table A171, the real table A151, the real table B152, and the real table C153, and a virtual table B221 consists of the data which application 22 needs. The configuration of a virtual table B221 is shown in drawing 7 (d). A virtual table B221 consists of a part number, a product name, a receipt number, and the contents of a failure. To the part number of the real table A151 corresponding to the part number of a virtual table A171, a part number To the product name of the real table A151 corresponding to the part number of a virtual table A171, a product name The receipt number supports the contents of a failure of the real table C153 corresponding to the "part number + branch number A+ branch number B" of a virtual table A171 in the contents of a failure at the receipt number of the real table B152 corresponding to the "part number + branch number A" of a virtual table A171, respectively.

[0065] Actuation of the gestalt of operation of the 2nd of this invention is explained to details with reference to drawing 6 - drawing 9.

[0066] drawing where drawing 8 explains actuation of the gestalt of the 2nd operation -- it is -- (a) -- the example of the real table A151 -- (b) -- the example of the real table B152 -- in (c), (d) shows the example of a virtual table A171, and (e) shows the example of a virtual table B221 for the example of the real table C153. In (b), although receipt number "002" and "003" were once received, since they were deleted after that, they are indicated to be "DEL" to deletion. Drawing 9 is drawing showing the flow of actuation of the gestalt of the 2nd operation.

[0067] When drawing 9 is referred to, application 22 The part number of the real table A151 and the part number of a virtual table A171 are connected. The "part number + branch number A" of the real table B152 and the "part number + branch number A" of a virtual table A171 are connected. Connect the "part number + branch number A+ branch number B" of the real table C153, and the "part number + branch number A+ branch number B" of a virtual table A171, and the virtual table B221 which consists of a part number, a product name, a receipt number, and the contents of a failure is defined. Access to RDB15 is required of RDBMS16 (steps A21-A22).

[0068] It judges whether RDBMS16 needs starting activation of a stored procedure 17, when the virtual table A171 analyzes the access request to RDB15 from application 22 (step M21), and a stored procedure 17 carries out [the virtual table] definition generation is specified, and when it has recognized that a stored procedure 17 needs to be starting performed, a stored procedure 17 is started (steps M22-M24). When starting activation of the specified stored procedure 17 has already been carried out and renewal of RDB15 is not made after that, a stored procedure 17 is not started. In this case, the processing result of the already performed stored procedure 17 is used.

[0069] The started stored procedure 17 performs the following processings.

(1) the real table C153 -- "-- a branch number B chooses the line containing the greatest thing as every part number + branch number A" within the same "part number + branch number A" (step S21). The line to which O mark is attached is chosen as the right end of drawing 8 (c) in this example.

(2) Connect the "part number + branch number A" of the real table B152, and the "part number + branch number A" as a result of the above (1) (step S22). At this time, the "part number + branch number A" of the real table B152 with which "DEL" is set as deletion of the real table B152 is removed. In this example, that remove and receipt number "002" and the line of "003" are connected since "DEL" is set as deletion is a line to which drawing 8 (b) and the right end of (c) have O mark.

(3) Create the data set of the virtual table A171 which consists of the part number of the real table A151, the branch number A as a result of the above (2), and the branch number B as a result of the above (2) (step S23). The output of a stored procedure 17 is shown in drawing 8 (d).

[0070] Then, RDBMS16 performs processing based on a virtual table B221 (step M25). The processing result of RDBMS16 is shown in drawing 8 (e).

[0071] Then, RDBMS16 returns a processing result to application 22 (step M26).

[0072] Application 22 performs reception and an application process for a processing result from RDBMS16 (step A23).

[0073] Thus, application 22 can obtain the newest data of RDB15 for which it is asking at a high speed by using the virtual table A171 which is the output of a stored procedure 17 instead of an index.

Namely, application 22 can be processed by acquiring the newest receipt number and the newest contents of a failure over a part number, without carrying out complicated processing.

[0074] In the gestalt of the operation by this invention mentioned above, it uses in order to memorize the program for performing processing actuation of a data base access method etc. to storage (not shown), such as a magnetic disk and an optical disk, as data, to read the memorized data and to operate a data base access method. Thus, the data which operates the data base access method by this invention is stored in a storage, and the function of a data base access method can be realized now by installing this storage.

[0075]

[Effect of the Invention] The 1st effect is only referring to a required stored procedure and always being able to refer to the newest data at a high speed from application. The reason is because a means which constitutes the function which generates the index for applications in a stored procedure was established.

[0076] The 2nd effect is that the load of a client is mitigated and network traffic also decreases. Since a stored procedure is performed by the server side, the reason is for there being also few exchanges which do not need to publish complicated SQL from application, can build processing only by application describing the data manipulation processing based on the table of a meaning key, and are generated between the application of a client and RDBMS of a server, and ending.

[0077] The 3rd effect is that special restoration treatment called re-creation of an index table physical for the index for applications is not needed at the time of the failure of a data base. The reason is because the stored procedure is used as a table with the imagination index for applications.

[Translation done.]

*** NOTICES ***

Japan Patent Office is not responsible for any damages caused by the use of this translation.

1. This document has been translated by computer. So the translation may not reflect the original precisely.
2. **** shows the word which can not be translated.
3. In the drawings, any words are not translated.

CLAIMS

[Claim(s)]

[Claim 1] A data base access method characterized by providing the following A data base which carries out storage are recording of the data A database management system which receives a demand from a data base access request means, accesses said data base, and returns an access result to said data base access request means A stored procedure activation means to create a table of a meaning key which was contained in said database management system, performed a stored procedure, and was defined as a stored procedure A data base access request means to give a demand which accesses said data base with reference to a table of a meaning key which said stored procedure activation means created to said database management system

[Claim 2] A client/server system which is characterized by providing the following and by which a client is connected with a server through a network Said server is a relational database which carries out storage are recording of the data, and consists of one or more real tables. A relational database management system which accesses said relational database based on processing demanded from application, and returns a processing result to said application A stored procedure which defines a virtual table connected with said real table, and creates a table of a meaning key It is the application which is equipped with a stored procedure starting means to start said stored procedure, publishes a demand which said client specifies a virtual table which said stored procedure defines, and accesses said relational database to said relational database management system, and receives a demand result from said relational database management system.

[Claim 3] Said stored procedure starting means is a data base access method according to claim 2 which starting activation of said stored procedure is already carried out, and is characterized by not starting said stored procedure when renewal of said relational database is not made after that.

[Claim 4] A virtual table which said stored procedure defines is a data base access method according to claim 2 characterized by carrying out a duty of an index when application accesses said relational database.

[Claim 5] It is the method of accessing a data base in a client/server system by which a client is connected with a server through a network. Application defines a table which connected a virtual table defined by a real table and stored procedure, and performs an access request to a relational database to a relational database management system. Said relational database management system analyzes an access request from said application. A virtual table which carried out starting activation of the stored procedure, and was defined as a stored procedure when a virtual table said stored procedure carries out [a virtual table] definition generation was specified is generated. It is the data base access method which processes an access request of application which used said virtual table, returns a processing result to application, and is characterized by application receiving a processing result from a relational database management system.

[Claim 6] Said stored procedure is the data base access method according to claim 5 characterized by being the processing which generates a virtual table which consists of a meaning key of the newest

hysteresis from a real table of the master data, and two real tables or more holding the hysteresis.
[Claim 7] Access request processing which defines a table with which application connected a virtual table defined by a real table and stored procedure, and performs an access request to a relational database to a rational database management system, Said rational database management system analyzes an access request from said application. Stored procedure starting processing in which said stored procedure is started when a virtual table said stored procedure carries out [a virtual table] definition generation is specified, Virtual table generation processing which generates a virtual table which performed said stored procedure and was defined as a stored procedure, Access processing which processes an access request of application which used said virtual table, and returns a processing result to application, A record medium characterized by recording a program for making a computer perform data receipt processing in which application receives a processing result from said rational database management system.

[Translation done.]

PTO 04-1710

Japanese Kokai Patent Application
No. 2001-5704

DATABASE ACCESS SYSTEM, METHOD, AND STORAGE MEDIUM

Sunao Koderu

*Seems to apply
cl. 101*

UNITED STATES PATENT AND TRADEMARK OFFICE
WASHINGTON, D.C. FEBRUARY 2004
TRANSLATED BY THE RALPH MCELROY TRANSLATION COMPANY

JAPANESE PATENT OFFICE
PATENT JOURNAL
KOKAI PATENT APPLICATION NO. 2001-5704

Int. Cl. ⁷ :	G 06 F 12/00 17/30
Identification Code:	513
Theme Codes (References):	5B075 5B082
Filing No.:	Hei 11[1999]-172028
Filing Date:	June 18, 1999
Publication Date:	January 12, 2001
Examination Request:	Requested
No. of Claims:	7 (Total of 11 pages)

DATABASE ACCESS SYSTEM, METHOD, AND STORAGE MEDIUM
[Deetabeesu akusesu hoshiki, hoho oyobi kioku baitai]

Inventor:	Sunao Koderu
Applicant:	000211329 Chugoku NEC Software, Ltd.
F Terms (References):	5B075 NK02 NK10 NK24 NK54 PQ02 PQ05 5B082 BA09 EA05 GA08

[There are no amendments to this patent.]

*

*

*

Claims

1. A database access system characterized in that it is equipped with a database for storing data, a database management system which gains access to the aforementioned database upon receiving a request from a database access request means and returns the access results to the aforementioned database access request means, a stored procedure execution means contained in the aforementioned database management system which executes a stored procedure in order to generate a table of unique keys defined by the stored procedure, and a database access request means which issues a request to gain access to the aforementioned database to the aforementioned database management system in reference to the table of unique keys generated by the aforementioned stored procedure execution means.

2. A database access system characterized in that in a client server system in which a server and clients are connected via a network, the aforementioned server is equipped with a relational database which comprises more than 1 base tables containing data, a relational database management system which gains access to the aforementioned relational database according to a processing requested by an application program and returns the access results to the aforementioned application program, a stored procedure which defines a virtual table correlated with the aforementioned base tables in order to generate a table of unique keys, and a stored procedure activation means which activates the aforementioned stored procedure; and the aforementioned client is provided with an application program used to specify the virtual table defined by the aforementioned stored procedure in order to issue a request to gain access to the aforementioned relational database and to the aforementioned relational database management system and receive the results of the request from the aforementioned relational database management system.

3. A database access method described under Claim 2 characterized in that the aforementioned stored procedure activation means does not activate the aforementioned stored procedure when the aforementioned stored procedure was already activated and executed, and the aforementioned relational database has not been renewed since then.

4. The database access method described under Claim 2 characterized in that the virtual table defined by the aforementioned stored procedure serves the role of indexes used when the application program gains access to the aforementioned relational database.

5. A database access method characterized in that it is a method for gaining access to a database in a client server system in which a server and clients are connected via a network; wherein, an application program defines a table, in which base tables and a virtual table defined by a stored procedure are correlated, and issues a request to gain access to a relational database to

* [Numbers in the margin indicate pagination in the foreign text.]

a relational database management system; the aforementioned relational database management system analyzes the access request from the aforementioned application program, activates/executes the stored procedure if the virtual table defined and generated in accordance with the aforementioned stored procedure is specified in order to generate the virtual table defined by the stored procedure, processes the access request from the application program which used the aforementioned virtual table, and returns the processing results to the application program; and the application program receives the processing results from the relational database management system.

6. The database access method described under Claim 5 characterized in that the aforementioned stored procedure refers to processing for generating a virtual table comprising unique keys for the latest history from 2 or more base tables comprising [at least] a base table for basic data and a base table for keeping its history.

7. A storage medium characterized in that it stores a program in order for a computer to execute access request processing during which an application program defines a table in which base tables and a virtual table defined by a stored procedure are correlated in order to issue a request to gain access to a relational database to a database management system, stored procedure activation processing during which the aforementioned database management system analyzes the access request from the aforementioned application program and activates the aforementioned stored procedure if the virtual table defined and generated in accordance with the aforementioned stored procedure is specified, virtual table generation processing during which the aforementioned stored procedure is executed in order to generate the virtual table defined by the stored procedure, access processing during which the access request from the application program which used the aforementioned virtual table is processed, and the processing results are returned to the application program, and data reception processing during which the application program receives the processing results from the aforementioned relational database management system.

Detailed explanation of the invention

0001

Technical field of the invention

The present invention pertains to a database access system in which access is gained to a database using a database management system and a method and a medium to this end. In particular, it pertains to a database access system in which access is gained to a database using a stored procedure registered to a database management system and a method and a medium to this end.

[0002]

Prior art

In the case of a relational database management system (will be abbreviated as RDBMS, hereinafter) as one type of database management system, the relational database (will be abbreviated as RDB, hereinafter) is managed in the form of tables. The tables include base tables which contain actual data and a virtual table (referred to as view also) which is generated from the data in an original base table at the time of the execution of an instruction but not used for storing data. An application program obtains necessary data in the RDB via the RDBMS using the SQL language used for making references to the tables.

[0003]

In a client/server system (will be abbreviated as C/S system, hereinafter), the server is equipped with an RDBMS and connected to an RDB, and an application program provided at the client's side issues an RDB access request to the RDBMS using the SQL language and receives the access results from the RDBMS.

[0004]

That is, in order for the application program of the client to retrieve necessary data, the application program issues a search condition formula in accordance with the SQL language format to the RDBMS of the server; and the RDBMS runs a data search in the RDB, selects data which match the search condition specified, and returns the search results to the application program of the client. At this time, a high-speed access technique, for example, generation of indexes for the search, is used by the RDBMS.

/3

[0005]

However, the indexes generated by the RDBMS are not always optimized for the application program.

[0006]

For example, an application program used for obtaining the latest prices of respective merchandises in an RDB comprising base table A and base table B shown in Figure 10 (a) and (b) will be examined.

[0007]

The application program carries out the following processing.

(1) It selects the lines showing the latest time stamps [literally; "date and time"] for those with the same merchandise numbers in base table B.

(2) It correlates base table A with the results of (1) above for the respective merchandise numbers.

(3) It generates a table comprising the merchandise numbers, merchandise names, and prices according to the correlation obtained in (2) above.

(4) It gains access to the table obtained as a result of (3) above in order to obtain data on the merchandise numbers, the merchandise names, and the prices.

[0008]

In this case, although indexes for the time stamps are generated by the RDBMS, no indexes for those with the latest time stamps among those with the same merchandises are generated. Thus, the program application must describe a processing procedure for selecting those with the latest time stamps among those with the same merchandise numbers in (1) above. As such, the indexes generated by the RDBMS are not optimized for an individual application program.

[0009]

In addition, because the aforementioned processing is executed by individual clients, request information from the respective clients and result information from the server are exchanged between the server and the clients in a complex manner via the network.

[0010]

Problems to be solved by the invention

In the aforementioned prior art, the indexes generated by the RDBMS are simple indexes based on data entries, and particular data needed by the application program are not generated in the index format, resulting in the application program becoming complicated. As a result, it creates another problem that the traffic between the RDBMS and the application program becomes busier.

[0011]

The purpose of the present invention is to present a means by which indexes needed by each application program are generated using a stored procedure registered to an RDBMS in

order to simplify the application program, so that the overall performance of the system can be improved.

[0012]

Means to solve the problems

The database access system as a first invention of the present patent application is equipped with a database for storing data, a database management system which gains access to the aforementioned database upon receiving a request from a database access request means and returns the access results to the aforementioned database access request means, a stored procedure execution means contained in the aforementioned database management system which executes a stored procedure in order to generate a table of unique keys defined by the stored procedure, and a database access request means which issues a request to gain access to the aforementioned database to the aforementioned database management system in reference to the table of unique keys generated by the aforementioned stored procedure execution means.

[0013]

In the case of the database access system as a second invention of the present patent application, in a client server system in which a server and clients are connected via a network, the aforementioned server is equipped with a relational database which comprises more than 1 base tables containing data, a relational database management system which gains access to the aforementioned relational database according to a processing requested by an application program and returns the access results to the aforementioned application program, a stored procedure which defines a virtual table correlated with the aforementioned base tables in order to generate a table of unique keys, and a stored procedure activation means which activates the aforementioned stored procedure; and the aforementioned client is provided with an application program used to specify the virtual table defined by the aforementioned stored procedure in order to issue a request to gain access to the aforementioned relational database to the aforementioned relational database management system and receive the results of the request from the aforementioned relational database management system.

[0014]

The database access system as a third invention of the present patent application is characterized in that in the second invention, the aforementioned stored procedure activation means does not activate the aforementioned stored procedure when the aforementioned stored procedure was already activated and executed, and the aforementioned relational database has not been renewed since then.

[0015]

The database access system as a fourth invention of the present patent application is characterized in that in the second invention, the virtual table defined by the aforementioned stored procedure serves the role of indexes used when the application program gains access to the aforementioned relational database.

[0016]

The database access method as a fifth invention of the present patent application is characterized in that it is a method for gaining access to a database in a client server system in which a server and clients are connected via a network; wherein, an application program defines a table, in which base tables and a virtual table defined by a stored procedure are correlated, and issues a request to gain access to a relational database to a relational database management system; the aforementioned relational database management system analyzes the access request from the aforementioned application program, activates/executes the stored procedure if the virtual table defined and generated in accordance with the aforementioned stored procedure is specified in order to generate the virtual table defined by the stored procedure, processes the access request from the application program which used the aforementioned virtual table, and returns the processing results to the application program; and the application program receives the processing results from the relational database management system.

[0017]

/4

The database access method as a sixth invention of the present patent application is characterized in that in the fifth invention, the aforementioned stored procedure refers to processing for generating a virtual table comprising unique keys for the latest history from 2 or more base tables comprising [at least] a base table for basic data and a base table for keeping its history.

[0018]

The storage medium as a seventh invention of the present patent application is characterized in that it stores a program in order for a computer to execute access request processing during which an application program defines a table in which base tables and a virtual table defined by a stored procedure are correlated in order to issue a request to gain access to a relational database to a database management system, stored procedure activation processing during which the aforementioned database management system analyzes the access request from the aforementioned application program and activates the aforementioned stored procedure if the virtual table defined and generated in accordance with the aforementioned stored procedure is

specified, virtual table generation processing during which the aforementioned stored procedure is executed in order to generate the virtual table defined by the stored procedure, access processing during which the access request from the application program which used the aforementioned virtual table is processed, and the processing results are returned to the application program, and data reception processing during which the application program receives the processing results from the aforementioned relational database management system.

[0019]

Embodiments of the invention

The database access system of the present invention will be explained in reference to Figure 1.

[0020]

In reference to Figure 1, the database access system of the present invention is configured with database 4, database management system 5, stored procedure execution means 6, and database access request means 7.

[0021]

Database 4 is a database for storing and accumulating data.

[0022]

Database management system 5 manages database 4, gains access to database 4 upon receiving a request from database access request means 7, and returns the access results to database access request means 7.

[0023]

Stored procedure execution means 6 executes a stored procedure registered to database management system 5. The stored procedure includes processing for generating a table of unique keys (indexes) used by database access request means 7.

[0024]

Database access request means 7 requests database management system 5 to activate stored procedure execution means 6 and gain access to database 4 using the execution result and receives the request results from database management system 5.

[0025]

Operations of the present invention will be explained. Here, assume that the stored procedure to be used by database access request means 7 has been registered to database management system 5 in advance using a well-known method.

[0026]

First, database access request means 7 specifies a table of unique keys (indexes) to be generated by the stored procedure and sends a request to gain access to database 4 to database management system 5.

[0027]

Database management system 5 lets stored procedure execution means 6 execute the specified stored procedure to generate the table of unique keys (indexes) to be used by database access request means 7, carries out a specified access request processing then, and returns the processing results to database access request means 7.

[0028]

Database access request means 7 receives the processing results from database management system 5.

[0029]

As described above, when the table of unique keys (indexes) used by database access request means 7 is generated by the stored procedure registered to database management system 5, database access request means 7 can obtain the data it needs from database 4 quickly.

[0030]

Embodiments of the present invention to which the database access system of the present invention is applied will be explained.

[0031]

A first embodiment of the present invention will be explained in detail in reference to figures. The first embodiment pertains to a system which runs a search in a RDB for managing the latest prices of merchandises using 2 base tables to obtain the latest prices of the merchandises for further processing.

[0032]

Figure 2 is a diagram illustrating the configuration of the first embodiment.

[0033]

In reference to Figure 2, in the first embodiment, server 1 and clients 2 are connected via network 3. Server 1 is equipped with RDBMS 12 and connected to RDB 11. RDBMS 12 contains stored procedure 13 and stored procedure activation means 14. Client 2 contains application program 21.

[0034]

RDB 11 is a relational database for storing/accumulating data, and comprises base table A 111 and base table B 112. Base table A 111 holds basic data, and base table B 112 keeps the history of the basic data.

[0035]

Base table A 111 is one of the tables which constitute RDB 11. Configuration of base table A 111 is shown in Figure 3 (a). Base table A 111 comprises merchandise numbers and merchandise names, and respective lines are distinguished from each other uniquely using the merchandise numbers as the primary keys.

[0036]

Base table B 112 is one of the tables which constitute RDB 11. Configuration of base table B 112 is shown in Figure 3 (b). Base table B 112 comprises merchandise numbers and time stamps and shows the latest prices of the merchandise. Every time a merchandise price is changed, a line showing a time stamp indicating the change of price and the new price is registered. The merchandise numbers correspond to the merchandise numbers in base table A 111, and base table A 111 and base table B 112 are correlated with each other as a result. The time stamps show the dates and the times those price are registered, and they show the order in which the prices of those with the same merchandise numbers are registered. Serial numbers may be used in place of the time stamps. The respective lines are distinguished from each other uniquely using the combination of a merchandise number and a time stamp (will be noted as "merchandise number + time stamp," hereinafter).

[0037]

RDBMS 12 manages RDB 11 and gains access to it. It processes a request issued from application program 21 and returns the processing results corresponding to the request issued to application program 21.

[0038]

Stored procedure 13 defines and generates virtual table A 131 based on base table A 111 and base table B 112 in order to select and generate a dataset. Stored procedure 13 is activated by stored procedure activation means 14. Name of virtual table A 131 is decided when stored procedure 13 is registered to RDBMS 12. Application program 21 can make reference to virtual table A 131 generated by stored procedure 13 by specifying said name. Furthermore, a well-known method is used as the method for registering stored procedure 13 to RDBMS 12.

[0039]

Virtual table A 131 is a virtual table comprising unique keys with the latest history. It is defined and generated by stored procedure 13 based on base table A 111 and base table B 112; wherein, selection conditions for combining the merchandise numbers in base table A 111 with the corresponding merchandise numbers in base table B 112 with the latest (greatest) time stamps are specified. Configuration of virtual table A 131 is shown in Figure 3 (c). Virtual table A 131 comprises merchandise numbers and time stamps; wherein, the merchandise numbers and the time stamps [sic] in base table A 111 correspond to the merchandise numbers and the latest (greatest) time stamps of those with the same merchandise numbers in base table B 112, respectively. From the viewpoint of application program 21, virtual table A 131 plays the role of indexes which constantly reflect the latest condition of RDB 11 when a reference is made from application program 21.

[0040]

Stored procedure activation means 14 activates stored procedure 13 when application program 21 sends a request to RDBMS 12 to use virtual table A 131 generated by stored procedure 13. If the specified stored procedure 13 was already activated and executed, and RDB 11 has not been renewed since then, stored procedure 13 is not activated. In this case, the processing results of stored procedure 13 already executed are used.

[0041]

Application program 21 defines virtual table B 211 correlated with virtual table A 131 defined and generated by stored procedure 13 and issues an access request to RDBMS 12 in

order to obtain the latest data from RDB 11. Processing results corresponding to the request issued are returned from RDBMS 12, and data processing is carried out based on the data obtained.

[0042]

Virtual table B 211 is a virtual table defined and generated by application program 21 based on virtual table A 131, base table A 111, and base table B 112, and it comprises the data needed by application program 21. Configuration of virtual table B 211 is shown in Figure 3 (d). Virtual table B 211 comprises merchandise numbers, merchandise names, and prices; wherein, the merchandise numbers, the merchandise names, and the prices correspond to the merchandise numbers in base table A 111 corresponding to the merchandise numbers in virtual table A 131, the merchandise names in base table A 111 corresponding to the merchandise numbers in virtual table A 131, the prices in base table B 112 corresponding to “merchandise number + time stamp” in virtual table A 131, respectively.

[0043]

Operations of the first embodiment of the present invention will be explained in detail in reference to Figure 1 through Figure 5.

[0044]

Figure 4 shows tables for illustrating the operations of the first embodiment; wherein, (a) shows a specific example of base table A 111, (b) shows a specific example of base table B 112, (c) shows a specific example of virtual table A 131, and (d) shows a specific example of virtual table B 211. From (a) and (b), it is clear that the latest price of the merchandise name “A A A” is “190” which is set at the time stamp “05261800.” Figure 5 is a flowchart illustrating the flow of the first embodiment.

[0045]

In reference to Figure 5, application program 21 correlates the merchandise numbers in base table A 111 with the merchandise numbers in virtual table A 131 as well as “merchandise number + time stamp” in base table B 112 with “merchandise number + time stamp” in virtual table A 131 to define virtual table B 211 comprising the merchandise numbers, the merchandise names, and the prices and requests access to RDB 11 to RDBMS 12 (Steps A11-A12).

[0046]

RDBMS 12 analyzes the RDB 11 access request from application program 21 (Step M11) so as to judge whether the activation/execution of stored procedure 13 is necessary or not if virtual table A 131 defined and generated by stored procedure 13 is specified, and it activates stored procedure 13 if it decide that the activation/execution of stored procedure 13 is necessary (Steps M12-M14). If the specified stored procedure 13 was already activated and executed, and RDB 11 has not been renewed since then, stored procedure 13 is not activated. In this case, the processing results of stored procedure 13 already executed are used.

[0047]

Activated stored procedure 13 executes the following processing.

(1) It selects the lines showing the latest time stamps for those with the same merchandise numbers for the respective merchandise numbers in base table B 112 (Step S11).

(2) It correlates the merchandise numbers in base table A 111 with the merchandise numbers obtained as a result of (1) above (Step S12).

(3) It generates a data set comprising the merchandise numbers in base table A 111 and the time stamps obtained as a result of (1) above for virtual table A 131 (Step S13). Resulting output of stored procedure 13 is shown in Figure 4 (c).

[0048]

Then, RDBMS 12 carries out processing based on virtual table B 211 (Step M15). Result of the processing by RDBMS 12 are shown in Figure 4 (d).

/6

[0049]

Subsequently, RDBMS 12 returns the processing results to application program 21 (Step M16).

[0050]

Upon receiving the processing results from RDBMS 12, application program 21 carries out the application processing (Step A13).

[0051]

As described above, application program 21 can obtain the latest data in RDB 11 it needs quickly using virtual table A 131 as the results outputted from stored procedure 13 in place of indexes.

[0052]

Next, a second embodiment of the present invention will be explained in detail in reference to figures. The second embodiment pertains to a system which runs a search in a RDB used to manage 3 base tables pertaining to product problem contents in order to obtain the latest product problem contents for further processing.

[0053]

Figure 6 is a diagram illustrating the configuration of the second embodiment.

[0054]

In reference to Figure 6, in the second embodiment, server 1 and clients 2 are connected via network 3, server 1 is equipped with RDBMS 16 containing stored procedure 17 and stored procedure activation means 14 and connected to RDB 15, and clients 2 are provided with application program 22; wherein, RDB 11, RDBMS 12, stored procedure 13, and application program 21 are replaced by RDB 15, RDBMS 16, stored procedure 17, and application program 22.

[0055]

Here, RDB 15, RDBMS 16, stored procedure 17, and application program 22 which are different from those of the first embodiment will be explained.

[0056]

RDB 15 is a relational database which stores and accumulates data, and it comprises base table A 151, base table B 152, and base table C 153. Base table A 151 is for holding basic data, and base table B 152 and base table C 153 are keeping the history of the basic data.

[0057]

Base table A 151 is one of the tables which constitute RDB 15. Configuration of base table A 151 is shown in Figure 7 (a). Base table A 151 comprises product numbers and product names, and the respective lines are distinguished from each other uniquely using the product numbers as the primary keys.

[0058]

Base table B 152 is one of the tables which constitute RDB 15. Configuration of base table B 152 is shown in Figure 7 (b). Base table B 152 comprises product numbers, branch numbers A, reception numbers, and deletions. The product numbers correspond to the product

numbers in base table A 151, and base table B 152 and base table A 151 are correlated with each other as a result. Branch numbers A indicate the order those with the same product numbers are received, and serial numbers or time stamps may be used to this end also. “Product number + branch number A” is used to distinguish the respective lines uniquely from each other. The reception numbers are serial numbers assigned throughout the products; and every time a product problem is received, a reception number is registered. Content of the problem is registered to base table C 153. If a new problem is received of the same product, its branch number A is incremented, and a new reception number is registered. Deletion indicates whether or not a reception number has been deleted after it was once received, and “DEL” is shown when the reception has been deleted.

[0059]

Base table C 153 is one of the tables which constitute RDB 15. Configuration of base table C 153 is shown in Figure 7 (c). Base table C 153 comprises product numbers, branch numbers A, branch numbers B, and problem contents. “Product number + branch number A” corresponds to “product number + branch number A” in base table B 152, and base table C 153 and base table B 152 are correlated with each other as a result. Branch number B indicates the order of the problem contents within “product number + branch A,” and serial numbers or time stamps may be used to this end also. The respective lines are distinguished from each other uniquely using “product number + branch number A + branch number B.” Problem contents corresponding to the reception numbers are registered as the problem contents. To renew the problem content pertaining to a reception number which has been received, branch number B is incremented when registering the renewed problem content in order to treat it as a different piece of data during the history management.

[0060]

RDBMS 16 manages RDB 15 and gains access to it. It processes a request issued from application program 22 and returns the processing results corresponding to the request issued to application program 22.

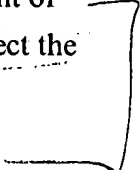
[0061]

Stored procedure 17 defines and generates virtual table A 171 based on base table A 151, base table B 152, and base table C 153 in order to select and generate a dataset. Stored procedure 17 is activated by stored procedure activation means 14. Name of virtual table A 171 is decided when stored procedure 17 is registered to RDBMS 16. Application program 22 can make reference to virtual table A 171 generated by stored procedure 17 by specifying said name.

Furthermore, a well-known method is used as the method for registering stored procedure 17 to RDBMS 16.

[0062]

Virtual table A 171 is a virtual table comprising unique keys with the latest history. It is defined and generated by stored procedure 17 based on base table A 151, base table B 152, and base table C 153; wherein, selection conditions for combining the highest branch number A within the products with the same product numbers in base table A 151 and base table B 152 with the highest branch number B within the same “product number + branch number A” in base table C 153 are specified. Configuration of virtual table A 171 is shown in Figure 7 (c) [sic]. Virtual table A 171 comprises product numbers, branch numbers A, and branch numbers B; wherein, the product numbers, branch numbers A, and branch numbers B correspond to the product numbers in base table A 151, the highest branch numbers A within the products with the same product number in base table B 152, and the highest branch numbers B within the same “product number + branch number A” in base table C 153, respectively. From the viewpoint of application program 22, virtual table A 171 plays the role of indexes which constantly reflect the latest condition of RDB 15 when a reference is made from application program 22.



[0063]

Application program 22 defines virtual table B 221 correlated with virtual table A 171 defined and generated by stored procedure 17 and issues an access request to RDBMS 16 in order to obtain the latest data from RDB 15. Processing results corresponding to the request issued are returned from RDBMS 16, and data processing is carried out based on the data obtained.

[0064]

Virtual table B 221 is a virtual table defined and generated by application program 22 based on virtual table A 171, base table A 151, base table B 152, and base table C 153, and it comprises the data needed by application program 22. Configuration of virtual table B 221 is shown in Figure 7 (d) [sic]. Virtual table B 221 comprises product numbers, product names, reception numbers, and problem contents; wherein, the product numbers, the product names, the reception numbers, and the problem contents correspond to the product numbers in base table A 151 which correspond to the product numbers in virtual table A 171, the product names in base table A 151 which correspond to the product numbers in virtual table A 171, the reception numbers in base table B 152 which correspond to “product number + branch number A” in

virtual table A 171, and the problem contents in base table C 153 which correspond to “product number + branch number A + branch number B” in virtual table A 171, respectively.

[0065]

Operations of the second embodiment of the present invention will be explained in detail in reference to Figure 6 through Figure 9.

[0066]

Figure 8 shows tables for illustrating the operations of the second embodiment; wherein, (a) shows a specific example of base table A 151, (b) shows a specific example of base table B 152, (c) shows a specific example of base table C 153, (d) shows a specific example of virtual table A 171, and (e) shows a specific example of virtual table B 221. In (b), reception numbers “002” and “003” have been deleted after they were once received, and “DEL” is shown under Deletion accordingly. Figure 9 is a flowchart illustrating the flow of the second embodiment.

[0067]

In reference to Figure 9, application program 22 correlates the product numbers in base table A 151 with the product numbers in virtual table A 171, “product number + branch number A” in base table B 152 with “product number + branch number A” in virtual table A 171, and “product number + branch number A + branch number B” in base table C 153 with “product number + branch number A + branch number B” in virtual table A 171, to define virtual table B 221 comprising the product numbers, the product names, the reception numbers, and the problem contents and requests access to RDB 15 to RDBMS 16 (Steps A21-A22).

[0068]

RDBMS 16 analyzes the RDB 15 access request from application program 22 (Step M21) so as to judge whether the activation/execution of stored procedure 17 is necessary or not if virtual table A 171 defined and generated by stored procedure 17 is specified, and it activates stored procedure 17 if it decides that the activation/execution of stored procedure 17 is necessary (Steps M22-M24). If the specified stored procedure 17 was already activated and executed, and RDB 15 has not been renewed since then, stored procedure 17 is not activated. In this case, the processing results of stored procedure 17 already executed are used.

[0069]

Activated stored procedure 17 executes the following processing.

(1) It selects the lines containing the highest branch numbers B within the same “product number + branch number A” for respective “product number + branch number A” (Step S21). In this example, the lines indicated by O marks to the right in Figure 8 (c) are selected.

(2) It correlates “product number + branch number A” in base table B 152 with “product number + branch number A” obtained as a result of (1) above (Step S22). At this time, “product number + branch number A” in base table B 152 for which “DEL” is set under Deletion in base table B 152 are excluded. In this example, because “DEL” is set for the lines for reception numbers “002” and “003,” they are excluded; and the lines attached with © to the right in Figure 8 (b) and (c) are correlated.

(3) It generates a dataset comprising the product numbers in base table A 151, branch numbers A obtained as a result of (2) above, and branch numbers B obtained as a result of (2) above for virtual table A 171 (Step S23). Resulting output of stored procedure 17 is shown in Figure 8 (d).

[0070]

Then, RDBMS 16 carries out processing based on virtual table B 221 (Step M25). Result of the processing by RDBMS 16 are shown in Figure 8 (e).

[0071]

Subsequently, RDBMS 16 returns the processing results to application program 22 (Step M26).

[0072]

Upon receiving the processing results from RDBMS 16, application program 22 carries out the application processing (Step A23).

[0073]

As described above, application program 22 can obtain the latest data in RDB 15 it needs quickly using virtual table A 171 as the results outputted from stored procedure 17 in place of indexes. That is, application program 22 can obtain the latest reception numbers and problem contents pertaining to the product numbers for processing without carrying out complicated processing.

[0074]

In the aforementioned embodiments in accordance with the present invention, the program for executing the processing operations of the database access system is stored as data in a storage device (not illustrated), such as a magnetic disk or an optical disk, and the stored data are read to operate the database access system. When the data for operating the database access system of the present invention are stored in a storage medium, functions of the database access system can be realized as said storage medium is installed.

[0075]

Effects of the invention

A first effect is that the application program can make reference to the latest data at all times simply by making reference to the necessary stored procedure. The reason is that a means for configuring the function to generate indexes for the application program is provided within the stored procedure.

/8

[0076]

A second effect is that burden on the clients can be reduced, and the network traffic is reduced at the same time. The reason is that because the stored procedure is executed by the server's side, there is no need for the application program to issue any complicated SQL, the processing carried out by the application program can be configured simply by describing data operation processing based on a table of unique keys, and the application program at the client's side and the RDBMS at the server's side need to communicate with each other less frequently.

[0077]

A third effect is that in the event of a database problem, no special restoration measure, such as regeneration of a physical index table to provide indexes for the application program, is needed. The reason is that a stored procedure in the form of virtual tables is utilized as indexes for the application program.

Brief description of the figures

Figure 1 is a diagram for illustrating the present invention.

Figure 2 is a diagram for illustrating the configuration of a first embodiment.

Figure 3 are diagrams showing the configurations of (a) base table A, (b) base table B, (c) virtual table A, and (d) virtual table B.

Figure 4 are tables for explaining the operations of the first embodiment; namely, (a) base table A, (b) base table B, (c) virtual table A, and (d) virtual table B.

Figure 5 is a flowchart of the operation flow of the first embodiment.

Figure 6 is a diagram for illustrating the configuration of a second embodiment.

Figure 7 are diagrams showing the configurations of (a) base table A, (b) base table B, (c) base table C, (d) virtual table A, and (e) virtual table B.

Figure 8 are tables for explaining the operations of the second embodiment; namely, (a) base table A, (b) base table B, (c) base table C, (d) virtual table A, and (e) virtual table B.

Figure 9 is a flowchart of the operation flow of the second embodiment.

Figure 10 are tables for explaining the conventional operations; namely, (a) base table A, (b) base table B, and (c) result.

Explanation of the symbols

1	Server
2	Client
3	Network
4	Database
5	Database management system
6	Stored procedure execution means
7	Database access request means
11	RDB
12	RDBMS
13	Stored procedure
14	Stored procedure activation means
15	RDB
16	RDBMS
17	Stored procedure
21	Application program
22	Application program
111	Base table A
112	Base table B
131	Virtual table A
151	Base table A
152	Base table B
153	Base table C
171	Virtual table A
211	Virtual table B
221	Virtual table B

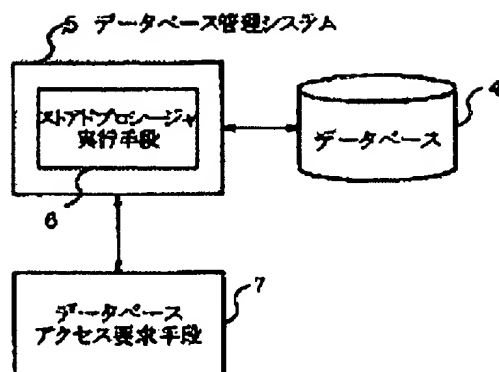


Figure 1

Key: 4 Database
 5 Database management system
 6 Stored procedure execution means
 7 Database access request means

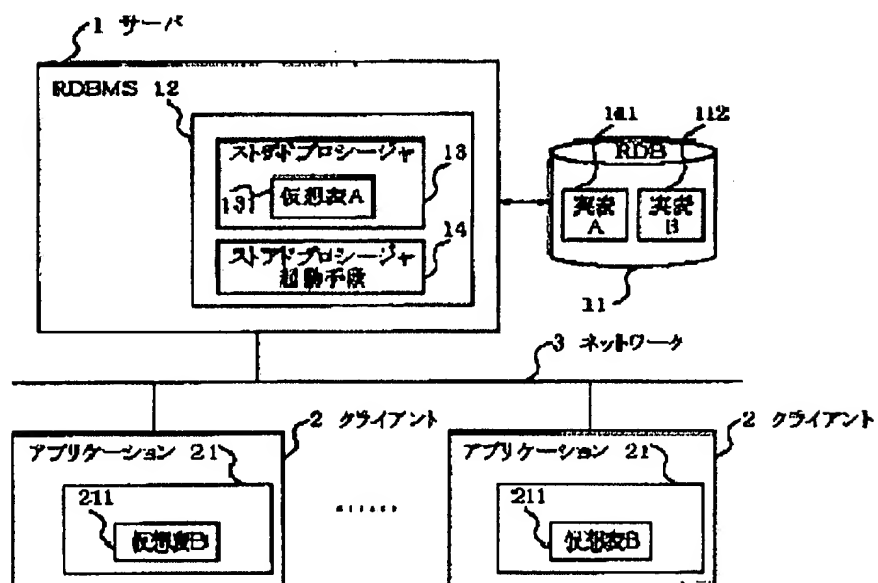


Figure 2

Key: 1 Server
 2 Client
 3 Network
 13 Stored procedure
 14 Stored procedure activation means
 21 Application program
 111 Base table A
 112 Base table B

131 Virtual table A
211 Virtual table B

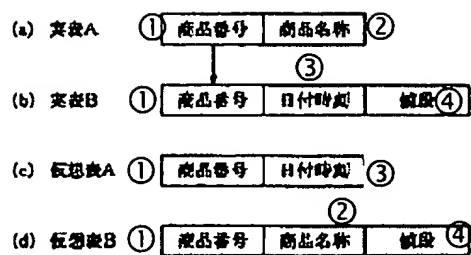


Figure 3

Key: (a) Base table A
(b) Base table B
(c) Virtual table A
(d) Virtual table B
1 Merchandise number
2 Merchandise name
3 Time stamp
4 Price

①	②		①	⑧	⑨
商品番号	商品名称		商品番号	日付時刻	値段
1	あああ	③	1	05261000	200
2	いはい	④	1	05261200	198
3	ううう	⑤	1	05261400	195
4	えええ	⑥	1	05261600	193
5	おおお	⑦	1	05261800	190
			2	05261000	100
			2	05261600	95
			3	05261000	498
			4	05261000	300
			4	05261200	295
			4	05261400	290
			5	05261000	98
			5	05261800	70

(a) 実表A

(b) 実表B

①	②		①	⑧	⑨
商品番号	日付時刻		商品番号	商品名称	値段
1	05261800		1	あああ ③	190
2	05261600		2	いはい ④	95
3	05261000		3	ううう ⑤	498
4	05261400		4	えええ ⑥	290
5	05261800		5	おおお ⑦	70

(c) 仮想表A

(d) 仮想表B

Figure 4

- Key:
- (a) Base table A
 - (b) Base table B
 - (c) Virtual table A
 - (d) Virtual table B
 - 1 Merchandise number
 - 2 Merchandise name
 - 3 A A A
 - 4 I I I
 - 5 U U U
 - 6 E E E
 - 7 O O O
 - 8 Time stamp
 - 9 Price

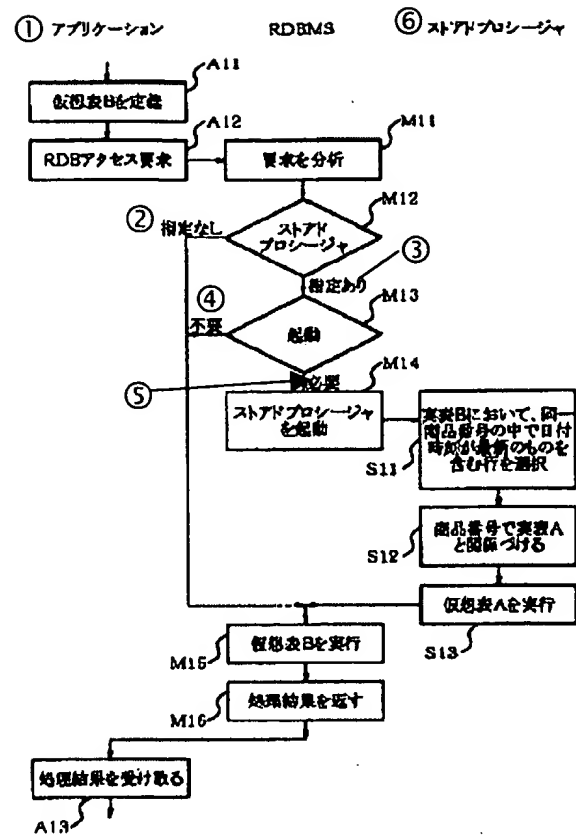


Figure 5

- Keys:
- 1 Application program
 - 2 Not specification
 - 3 Specified
 - 4 Not needed
 - 5 Needed
 - 6 Stored procedure
 - A11 Define virtual table B
 - A12 RDB access request
 - A13 Receive processing results
 - M11 Analyze request
 - M12 Stored procedure
 - M13 Activation
 - M14 Activate stored procedure
 - M15 Execute virtual table B
 - M16 Return processing results
 - S11 Select the lines containing the latest time stamps among those with the same merchandise numbers in base table B
 - S12 Correlate with base table A using merchandise numbers
 - S13 Execute virtual table A

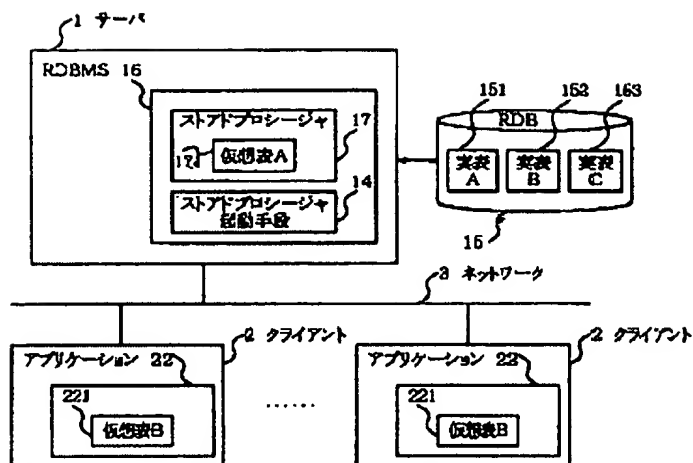


Figure 6

- Key:
- 1 Server
 - 2 Client
 - 3 Network
 - 14 Stored procedure activation means
 - 17 Stored procedure
 - 22 Application program
 - 151 Base table A
 - 152 Base table B
 - 153 Base table C
 - 171 Virtual table A
 - 221 Virtual table B

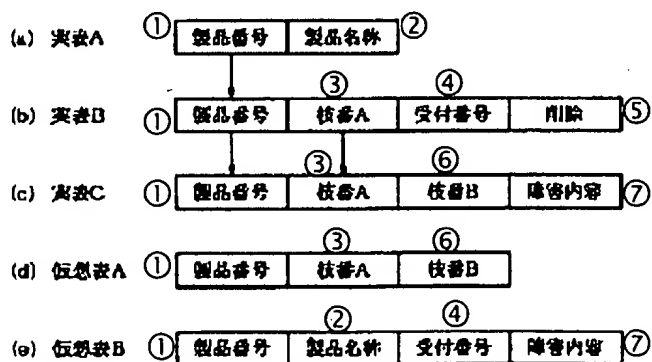


Figure 7

- Key:
- (a) Base table A
 - (b) Base table B
 - (c) Base table C
 - (d) Virtual table A
 - (e) Virtual table B
 - 1 Product number

- 2 Product name
 3 Branch number A
 4 Reception number
 5 Deletion
 6 Branch number B
 7 Problem content

①	②	①	⑥	⑦	④
製品番号	製品名称	製品番号	枝番A	受付番号	削除
1	あい	1	1	001	
2	かき	2	1	002	DEL
3	さし	2	2	004	
		3	1	003	DEL
		3	2	005	

(a) 実表A

①	⑥	⑨	⑩
製品番号	枝番A	枝番B	障害内容
1	1	1	A
1	1	2	B
2	1	1	C
2	2	1	D
3	1	1	E
3	2	1	F
3	2	2	G
3	2	3	H

(b) 実表B

①	⑥	⑨	①	②	⑦	⑩
製品番号	枝番A	枝番B	製品番号	製品名称	受付番号	障害内容
1	1	2	1	あい	001	B
2	2	1	2	かき	004	D
3	2	3	3	さし	005	H

(c) 実表C

(d) 仮想表A

(e) 仮想表B

Figure 8

- Key: (a) Base table A
 (b) Base table B
 (c) Base table C
 (d) Virtual table A
 (e) Virtual table B
 1 Product number
 2 Product name
 3 A I U
 4 Ka Ki Ku
 5 Sa shi Su
 6 Branch number A

- 7 Reception number
 8 Deletion
 9 Branch number B
 10 Problem content

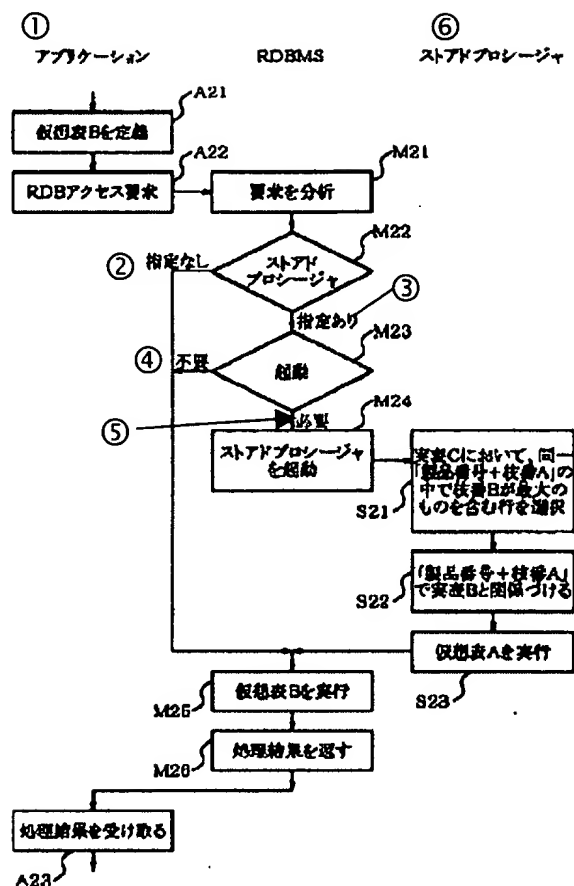


Figure 9

- Key:
- 1 Application program
 - 2 Not specification
 - 3 Specified
 - 4 Not needed
 - 5 Needed
 - 6 Stored procedure
 - A21 Define virtual table B
 - A22 RDB access request
 - A23 Receive processing result
 - M21 Analyze request
 - M22 Stored procedure
 - M23 Activation
 - M24 Activate stored procedure
 - M25 Execute virtual table B
 - M26 Return processing result

- S21 Select the lines containing the latest time stamps among those with the same
 “merchandise number + branch number A” in base table C
 S22 Correlate with base table B using “merchandise number + branch number A”
 S23 Execute virtual table A

①	②	①	⑧	⑨
商品番号	商品名称	商品番号	日付時刻	値段
1	あああ③	1	05261000	200
2	いはい④	1	05261200	198
3	ううう⑤	1	08261400	195
4	えええ⑥	1	05261600	193
5	おとお⑦	1	05261800	190
		2	05261000	100
		2	05261600	98
		3	05261000	498
		4	05261000	300
		4	05261200	298
		4	05261400	290
		5	05261000	98
		5	05261800	70

(a) 実表A

(b) 実表B

①	②	⑨
商品番号	商品名称	値段
1	あああ③	190
2	いはい④	98
3	ううう⑤	498
4	えええ⑥	290
5	おとお⑦	70

(c) 結果

Figure 10

- Key: (a) Base table A
 (b) Base table B
 (c) Result
 1 Merchandise number
 2 Merchandise name
 3 A A A
 4 I I I
 5 U U U
 6 E E E
 7 O O O
 8 Time stamp
 9 Price

Programming Techniques

R. M. McCURE, Editor

Policy Statement for Programming Techniques Department

Although the policies of a journal, such as Communications of the ACM, can be determined in part from the selection of papers that are printed, it appears that a formal policy statement may be of assistance both to authors and readers.

The purpose of the Programming Techniques department is to advance the state of the art in programming methodology by publishing high quality papers that fit either of two classes. The first class consists of papers that put forth new (or little known) ideas in programming methodology, relate them to prior art and literature, and explain their merits and demerits. Papers in this class should extend the knowledge of the reader already in the field of programming. In the second class are those papers that are commonly called tutorials. To fit in this class, a paper should present a comprehensive look at a particular subject and its relevant literature. It should provide a perspective from which a novice can become informed about a clearly defined subject. Both types of papers must reference prior art, including careful citation of both open and manufacturers' literature. This is imperative if our profession is to overcome the unfortunate oral tradition that has plagued us.

In general, a paper reporting results that can be expressed as an algorithm should be put into that form and submitted to the Algorithms section unless the derivation of the algorithm has unusual interest. Finally, papers deriving their merit from the idiosyncrasies of a particular machine or software system can be published only under unusual circumstances; the various user groups provide a better means for reaching the proper audience. — R.M. McC.

Storage Organization in Programming Systems

JANE G. JODEIT
Rice University, Houston, Texas

The system of program and data representation that has been in use on the Rice University computer for five years is described. Each logical entity in storage occupies a block of consecutive memory locations. Each block is labeled by a codeword and may contain a program, a data vector, or codewords which in turn label blocks to form arrays. This storage arrangement is discussed with its realized advantages in programming systems: simplicity of programmed addressing, flexibility of data structures, efficiency of memory utilization, variability of system composition during execution, means of linkage between programs and from programs to data, and basis for storage protection. The application of labeled blocks may be extended to areas of time-sharing and multimedia storage control. On the basis of experience at Rice, some ideas on such extensions are presented.

KEY WORDS AND PHRASES: storage allocation, storage organization, storage control, codewords, data representation, program representation, data structures, storage protection, addressing mechanisms, paging, segmentation, file handling

CR CATEGORIES: 4.30, 4.40, 6.20

Introduction

In this paper a representation of data and programs in storage that contributes organizational simplicity, coding convenience, and functional versatility in programming systems is described. Here programming system means the realization of a problem solution on a computer, anything from mathematical analysis to language translation.

A problem solution is defined by a collection of entities, programs and data items specifically. The generic term for such an entity is an *array*. Each array is named and contains as elements data (which may be the instructions of a program) or subarrays. In a programming system for the Rice computer the elements of an array form a *block*, a set of consecutive memory locations which has been called a "segment" [3]. Each block is labeled by a *codeword*, a word which corresponds to the name of the array whose elements occupy the block. If A is an array, the i th element of A is designated (A, i) . If the elements of A are subarrays, the i th word in the block for A is a codeword which labels the array (A, i) .

Thus an array is a tree structure. The name is the source from which the elements branch. If the elements are arrays, they in turn branch; if the elements are data, they are terminal. A source of branches is represented by a codeword; the set of branches from a single source is

This work was supported in part by the US Atomic Energy Commission, Contract Number AT-(40-1)-2572 to the Rice Computer Project, Rice University, Houston, Texas.

represented by a block containing codewords or data as appropriate.

A codeword which corresponds to a simple name, as A above, but not (A, i) , is called a primary codeword. All subarrays and data elements of an array are addressed "relative" to the simple name. This just means that the m th element of the n th subarray in the array DATA is named $(DATA, n, m)$; it has no other designation. The set of primary codewords then completely catalogs the entities of a programming system and all addressing is done through these codewords. The operating system provides dynamic allocation of blocks and maintenance of codewords. Primary codewords never move, and the addressing is independent of system composition and storage allocation.

Codewords as Block Labels and Their Use in Addressing

A set of consecutive storage locations is called a memory block. Every such block is labeled by a single word called a codeword. The codeword for a block corresponds to the name of the block; it contains descriptive information about the block, and a portion of the codeword is used in indirectly addressing the block content.

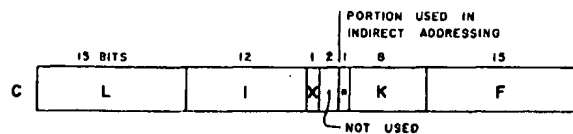


FIG. 1

As realized on the Rice University computer the general codeword format is shown in Figure 1, where

L is the length of the block labeled by the codeword C ;

I is the relative address of the first word in the block labeled by C ;

X is on if the block labeled by C contains codewords;

$*$ is on if indirect addressing is to be iterated into a word in the block labeled by C ;

K is present if the block labeled by C is indexed; i.e. if individual words in the block are to be addressed from outside the block, K then specifies the index register used to give the relative address of a word in the block (data vectors are indexed, programs are not);

F is an address associated with the block labeled by C so that the address of the first word in the block is $F' = F + I$.

The portion of a codeword used in indirect addressing is designed to be used with the hardware definition of the Rice computer. Indirect addressing may be iterated any number of times, and indexing by any of eight registers may be specified for each step. If C^i is the codeword in use at the i th level of indirect addressing, the hardware obtains $*^i$, K^i , and F^i from C^i and proceeds as follows:

- (1) If K^i is present, use contents of register specified and add to obtain $C^{i+1} = (K^i) + F^i$.

If K^i is not present, $C^{i+1} = F^i$.

- (2) If $*^i$ is on, return to step (1) for codeword C^{i+1} at level $i + 1$.
If $*^i$ is not on, use C^{i+1} as final address and do not iterate.

The initiation of indirect addressing is from an instruction, say at C^0 , which contains in its indirect addressing portion $*^0$, F^0 , and perhaps K^0 . Thus, from a single instruction the codeword address C^1 is determined and the hardware iterates through the indirect addressing procedure to provide the final address.

The full generality of codewords can be implemented with maximum efficiency with such hardware. It is surprising that more computers do not employ such an indirect addressing definition or some equivalent addressing mechanism. With more restrictive hardware capabilities the full generality of a codeword system can be realized at the expense of some efficiency, or some generality can be sacrificed and the most common applications handled efficiently.

Block Content and Addressing

Given the codeword definition of the previous section, we now examine how labeled blocks are used to build the elements of a programming system.

In general, any "named" item in the codeword system is called an *array*. On the highest level, that addressed in code, is a single codeword which corresponds to the name of the array and labels a block which may contain codewords. On the lowest level is the *data* of the array. The intermediate levels are formed by blocks of codewords, the *structure* of the array. The array forms most frequently encountered are discussed in detail later.

A *program* P may be considered as a set of words to be executed as instructions and should, for efficient control hardware utilization, occupy consecutive storage locations. Thus a program P occupies a memory block. Assume a single entry point to P ; then the block for P need not be indexed because only one word need be addressed from other programs. If P is of length k words with p words of linkage information, the program and its codeword appear as shown in Figure 2. Control is transferred to program P by the single operation:

transfer control to $*C_P$

where $*$ specifies indirect addressing through C_P , the codeword for P . A single step of indirect addressing is performed:

$$*C_P \xrightarrow{\text{indirect addressing}} F + p$$

and the final address obtained is $F + p$, the address of the first word of code for program P . The address $F + p$ never appears in code, *only* in the codeword for P . The address C_P which appears in *all* coded references to the program is invariant, while the address F may vary from run to run, or even within a run, as a function of total storage requirements.

A *vector* V may be considered as a set of words addressed by their relative position in the set and should, for efficient index hardware utilization, occupy consecutive storage

locations. Thus the vector V occupies an indexed memory block. If V is of length n words with the first word at relative position 1 and if register i is to be used for indexing,

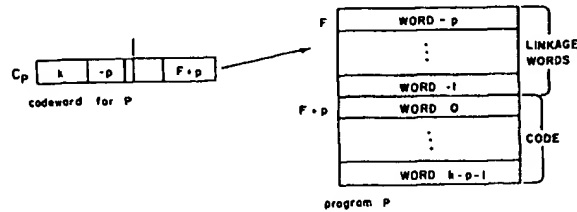


FIG. 2

the vector and its codeword appear as shown in Figure 3. Access to the p th element of vector V is accomplished by the two operations:

- (1) set index register i to p
- (2) access $*C_V$

where $*$ specifies indirect addressing through C_V , the codeword for V . A single step of indirect addressing is performed in step (2):

$$*C_V \xrightarrow{\text{indirect addressing}} p + F - 1$$

and the final address obtained is that of the element V_p , the p th word in the block beginning at location F . Again,

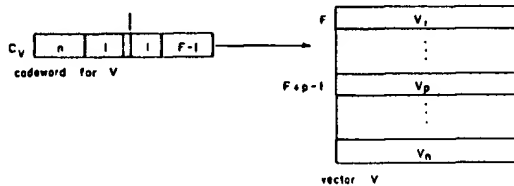


FIG. 3

the address $F-1$ never appears in code, *only* in the codeword for V . Code which references V is dependent only on the invariant codeword address, never on the physical location of the memory block for the vector.

A two-dimensional data structure, *matrix* M , may be realized as a vector of vectors. If the matrix M is m rows by n columns in size, then it will be represented as a vector of m vectors each n words in length. Thus the matrix M occupies m indexed memory blocks (one per row) of n data words each, and the codewords for the rows occupy an

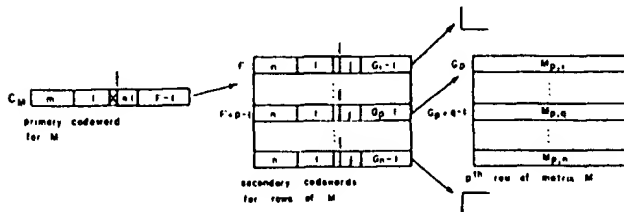


FIG. 4

indexed memory block of m codewords. If the "upper left" element of matrix M is to be element $M_{1,1}$ and row and column indices are to be specified in registers i and j re-

spectively, the matrix structure appears as shown in Figure 4. Access to the q th element of the p th row of matrix M is accomplished by the three operations:

- (1) set index register i to p
- (2) set index register j to q
- (3) access $*C_M$

where $*$ specifies indirect addressing through C_M , the codeword for M . Two steps of indirect addressing are performed in step (3):

$$*C_M \xrightarrow{\text{indirect addressing}} *(p + F - 1) \xrightarrow{\text{indirect addressing}} q + G_p - 1$$

and the final address obtained is that of the element $M_{p,q}$, the q th word in the block beginning at location G_p , which is addressed from the p th word in the block beginning at location F . The physical locations of the blocks which form the matrix never appear in code, *only* in codewords. Code which references elements of M is dependent only on the invariant highest level (primary) codeword address. Another very important point is that the code for access to matrix M in no way depends on the lengths m and n , only on the fact that M is two-dimensional. Hence, while the location of blocks which include M may vary as a function of total storage requirements, the size parameters m and n may as easily vary as a function of dynamic problem definition.

In general, array definition is extremely flexible in the codeword system, so this organizational form lends itself naturally to a large variety of computer problem descriptions.

If A is an array, the elements A_i are all data or all arrays. If A_i are data, A is one-dimensional (as programs and vectors described earlier). If A_i are arrays, they are just subarrays of A ; any array A_i may be defined or undefined at any time. The dimension and size of any A_i is independent of all others. The same rules of definition apply for arrays A_i as for A .

Thus a matrix may have rows of unequal length, as in the case of a triangular matrix, or only a subset of its rows defined at any time. An array of programs may be defined. This has been useful in compilation at Rice where on the basis of three integer values a code-generating routine is selected; not all triads are meaningful, so the array of code-generating programs is sparse. Programs may be inserted when new triads are defined, and any program may be modified and replaced without effect on others.

Codeword Location and Reference by Programs

The organization of the codeword system provides parallel tables with one entry per named item:

- *symbol table* (ST) containing names of items, and
- *value table* (VT) containing values of scalars and codewords for nonscalars (arrays).

The address of the VT entry for an item named A will be denoted VT_A . If A is a scalar, it is addressed at VT_A during execution. If A is a nonscalar, it is addressed indirectly through its codeword at VT_A . The location of

VT and the order of VT entries is a function of system composition. So a coded reference to an item named A is made indirectly through a linkage word L_A in the program:

transfer control to $\bullet L_A$

or

access $\bullet L_A$

Loading of the program provides the address VT_A in the linkage word L_A ; \bullet is on in L_A only if A is a nonscalar. The first indirect addressing operation then provides:

$$\bullet L_A \xrightarrow[\text{addressing}]{\text{indirect}} \begin{cases} VT_A, & \text{for scalar } A \\ \bullet VT_A, & \text{for nonscalar } A \end{cases}$$

Subsequent addressing is just as if VT_A had been addressed initially.

These linkages are illustrated in Figure 5 by the program P which references scalar $SCALR$ and nonscalar $ARRAY$.

The linkages discussed thus far have been for *fixed references*, the name for an item being known at coding time. Programs may also reference parameters which take on value assignments at each execution. Linkages to parameters are for *variable references*.

Parameters are provided for a program on a pushdown list W . The first free space in W is maintained during execution as a pointer WP . A parameter reference is coded indirectly through a linkage word located in W at a fixed position relative to the value of WP upon entry to the program. Index register P is set to the value of WP initially in each execution, and the linkage word for parameter A is located in W at $W_A = (P) + P_A$ where P_A is constant for all executions. Program reference to parameter A is accomplished by indirect addressing through W_A :

transfer control to $\bullet((P) + P_A)$

or

access $\bullet((P) + P_A)$

which may be written

transfer control to $\bullet W_A$

or

access $\bullet W_A$

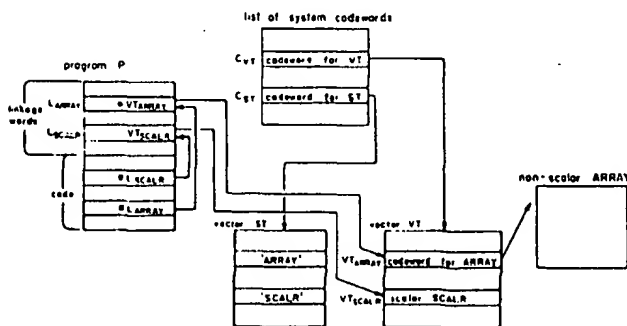


FIG. 5

Linkage for parameters and content of linkage words in W are illustrated in Figure 6 by program R which provides scalar $SCALR$ and nonscalar $ARRAY$ as parameters to program Q .

One further case must be considered. If PAR is a parameter in program R and R must execute Q with PAR as a parameter, R very simply copies its linkage word W_{PAR} into W in the list of parameter linkages prepared for Q . Then parameters may be passed to any level of program nesting during execution.

Dynamic Storage Allocation

The memory configuration for dynamic allocation in the codeword system consists of

- first, the control area which contains special machine registers, manual communication region, and the list of system codewords;

- second, any memory blocks which are not to be dynamically allocated—as the elements of the operating system;

- third, the remainder of the memory as the dynamic storage allocation domain.

Dynamic allocation in memory is defined by the two basic procedures:

- activation*, or creation, of a memory block labeled by a codeword, and

- inactivation* of a memory block labeled by a codeword so that the space may be subsequently used in allocation for other blocks.

Initial loading of programs and data is just a sequence of activations, and the blocks will be sequentially located in the storage domain. As a run progresses, blocks may be inactivated and new ones activated, so the general state of the storage domain is a mixture of active and inactive blocks.

Each *active block* in the storage domain is labeled by a codeword, which may itself be a word in an active block of codewords. Each active block is headed by a *back-reference word* which contains the codeword address for the block.

Each *inactive block* in the storage domain contains in its first word its length. One inactive block is used as the *source* of space for activations. This source is initially the

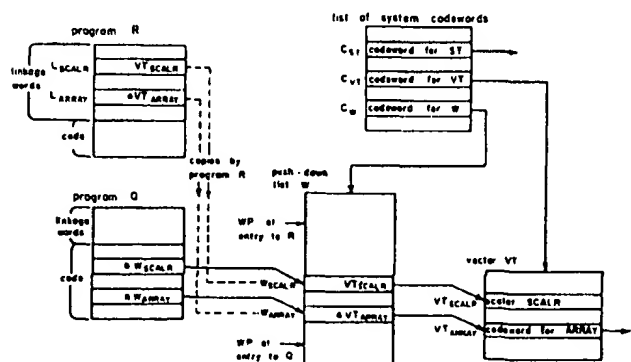


FIG. 6

whole domain. When the source is exhausted, active blocks are packed at one end of the domain, and the resulting single inactive block is designated as the source. This packing procedure is called *reorganization*. The memory of the Rice computer is not paged; if it were, reorganization would be effected by packing the page table [1, 4]. Notice that with paging, some storage economy is sacrificed since two blocks should not occupy the same page.

Each dynamic allocation request is specified by a codeword address and the allocation *operation* to be performed on the block is labeled by the codeword, either to free the block or to take a space of length n words.

The *freeing* of space labeled by a given codeword is performed by recursive inactivation of all blocks in the array labeled. Each block inactivated has its codeword cleared to signify that it no longer labels an active block.

The *taking* of a block of n words to be labeled by a given codeword is performed by first freeing the array labeled (if any) and then obtaining an active block $n+1$ words long (including the back-reference word) in the domain. So, a new block definition automatically replaces an old one.

Operations on Arrays

There are many useful operations on arrays that are easily implemented in the codeword system. Already the storage control operations of block creation to form arrays and freeing of arrays have been mentioned.

Mathematical operations on data arrays are familiar, such as transposition of a matrix or cross-correlation of two vectors. A routine to perform such an operation receives the name of an operand, i.e. its codeword address, as an argument. The routine then has access to the codeword for the array as well as for the array elements. Information such as dimension, size, and natural array indices are available without being given explicitly as arguments.

The codeword system provides a file structure very much like that described for secondary storage organization in the MULTICS system [2]. The implementation on the Rice computer provides a representation in primary storage which is immediately applicable through a hierarchy of storage devices. The same information which facilitates addressing and system component linkage is used by the operating system for file handling functions such as input-output, execution, insertion and deletion, and establishment of paths to file elements. The same notations, or naming conventions, are used in the designation of file manipulations and the description of data processing. Also, the file-level operations may be carried out from the console, as an operation quite independent of program execution, or from a program as it runs.

Memory Protection

Interest in multiprogramming and time-sharing computer applications has focused considerable attention on the problem of memory protection [3, 4, 9]. The objective has been mainly to prevent each memory resident from interfering with all others. Codewords provide the basis

for a logical protection scheme, one that insures that no memory references violate the block definitions of the running system. This scheme differs from those which provide protection per page of memory. If vector V is defined to contain elements V_1, V_2, \dots, V_6 , logical protection will prevent reference to V_6 ; physical, or page, protection will prevent this reference only if the word after V_6 lies in a different page and that page is unallowed to the program generating the reference.

The first premise for logical protection with codewords is that all memory references from a program to blocks outside itself are through primary codewords. This is not an unreasonable requirement; it is not different from the requirement that separate entities be given distinct segment numbers [3, 4]. Then, for each codeword in the indirect addressing chain which labels an indexed block, the index value k is checked to see that $I \leq k < I + L$. I and L are given in the codeword and are the relative address of the first word in the block and the length of the block respectively. This checking can be implemented in the hardware and is planned for the Rice computer at no loss in memory speed. Logical protection is now implemented in software at Rice,¹ because it is slow, it is used only for debugging.

To prevent a user from using a codeword which labels an array which is not his, requires a notation in (or on) the codeword which has not been included in the earlier definitions. One bit would suffice; it would be maintained as execution switches from user to user because it would appear only on the small set of primary codewords for the user in control. Alternatively, a field could contain a user number which would not change while his system was resident in memory. Shared data would have a codeword for each user allowed access.

Extensions

The codeword system for the Rice computer provides organization and control of primary storage for a single user. The restrictions of this particular implementation are not imposed by inadequacies of the theory. The descriptive properties of codewords, the modularity of array storage, and the protection potential in the system allow the codeword storage organization to be applied in a multiprogramming environment. Interrupt logic in the hardware and adequate secondary storage media would be essential. Hardware features for codeword recognition and special actions due to particular codeword content are suggested.

The design of a codeword system to serve more than one memory user at a time would require that each user have his own table of codewords (*value table* described earlier). Each table would be an element of an array which would catalog the systems of all users. Shared entities would have

¹ The Rice computer hardware provides two tag bits per word [8] which are not part of the data content but are used for control. Codewords are "marked" with a tag value which causes a trap out of the indirect addressing chain to a service routine. The service routine checks the validity of index values on the basis of the content of codewords in the chain.

codewords in several tables, or a collection of users would be permitted access through some tables.

Immediately, the allocation of primary storage involves overlay and automatic retrieval from secondary storage. As in the B8500 system [9], codewords may be marked when the block labeled is not available; an interrupt would allow intervention for retrieval. When a block is not in memory its codeword may be used to designate where it is. Codewords may be used for the collection of usage statistics [5] to aid in the decision about what to overlay. Dynamic demand would determine which blocks were in memory at any time; not all arrays or all of any array for a running system need be present.

Structured arrays have been designed for secondary storage files [2]. This has been done at Rice with no representational difficulties, but only on magnetic tape, which is a poor medium for the application. It has been proposed at Rice that the device which controls transmission between primary and secondary storage would recognize codewords; it would set codeword and back-reference content to properly define an array in the storage to which it is being transmitted. Thus a single command would suffice to move an entire array to or from memory; buffering and processor control would be minimized.

RECEIVED AUGUST, 1967; REVISED AUGUST, 1968

McNaughton—cont'd from page 740

structing combinational and sequential nets out of truth-function (i.e. Boolean) gates with built-in delays. Decomposition of combinational and sequential machines.

SOME AVAILABLE TEXTS

1. ARBIB, M. *Brains, Machines and Mathematics*. McGraw-Hill, New York, 1964.
2. CALDWELL, S. *Switching Circuits and Logical Design*. Wiley, New York, 1958.
3. CHOMSKY, N. *Aspects of Linguistics*. M.I.T. Press, Cambridge, Mass., 1966.
4. DAVIS, MARTIN. *Computability and Unsolvability*. McGraw-Hill, New York, 1958.
5. — (Ed.). *The Undecidable: Basic Papers on Undecidable Problems and Computable Functions*. Raven, Hewlett, N.Y., 1965.
6. GILL, A. *Introduction to the Theory of Finite-State Machines*. McGraw-Hill, New York, 1962.
7. GINSBURG, S. *An Introduction to Mathematical Machine Theory*. Addison-Wesley, Reading, Mass., 1962.
8. —. *The Mathematical Theory of Context-Free Languages*. McGraw-Hill, New York, 1966.
9. GLUSHKOV, VIKTOR M. *Introduction to Cybernetics*. (Trans.). Academic Press, New York, 1966.
10. HARRISON, M. *Introduction to Switching and Automata Theory*. McGraw-Hill, New York, 1965.
11. HARTMANIS, J., AND STEARNS, R. E. *Algebraic Structure of Sequential Machines*. Prentice-Hall, Englewood Cliffs, N.J., 1966.
12. HENNIE, FREDERICK C. *Finite-State Models for Logical Machines*. Wiley, New York, 1968.

REFERENCES

1. COMFORT, WEBB T. A computer system design for user service. Proc. AFIPS 1965 Fall Joint Comput. Conf., Vol. 27, Pt. 1, Spartan Books, New York, pp. 619-626.
2. DALEY, R. C., AND NEUMANN, P. G. A general-purpose file structure for secondary storage. Proc. AFIPS 1965 Fall Joint Comput. Conf., Vol. 27, Pt. 1, Spartan Books, New York, pp. 213-229.
3. DENNIS, JACK B. Segmentation and the design of multiprogrammed computer systems. *J. ACM* 12, 4(Oct. 1965), 589-602.
4. GLASER, E. L., COULEUR, J. F., AND OLIVER, G. A. System design of a computer for time sharing applications. Proc. AFIPS 1965 Fall Joint Comput. Conf., Vol. 27, Pt. 1, Spartan Books, New York, pp. 197-202.
5. GRAHAM, MARTIN. Memory hierarchy study. Tech. Rep., 1965. U. of California, Computation Center, Berkeley, Calif.
6. ILIFFE, J. K. The use of the Genie system in numerical calculations. In *Annual Review in Automatic Programming*, Vol. 2. Pergamon Press, New York, 1961, p. 1.
7. —, AND JOEIT, JANE G. A dynamic storage allocation scheme. *Comput. J.* 5 (Oct. 1962), 200-209.
8. JOEIT, JANE G., AND SITTON, GARY A. Tags for description and control. Rep. ORO-2572-9, Rice U. Comput. Proj., Houston, Texas, Feb., 1967.
9. McCULLOUGH, JAMES D., SPEIERMAN, KERMITH, II., AND ZURCHER, FRANK W. A design for a multiple user multiprocessing system. Proc. AFIPS 1965 Fall Joint Comput. Conf., Vol. 27, Pt. 1, Spartan Books, New York, pp. 611-617.
13. HERMES, H. *Enumerability, Decidability and Computability: An Introduction to the Theory of Recursive Functions* (trans.). Academic Press, New York, 1965.
14. HUMPHREY, W. S. *Switching Circuits with Computer Applications*. McGraw-Hill, New York, 1958.
15. KNUTH, DONALD E. *The Art of Computer Programming*, Vol. 9, *Theory of Languages*. Addison-Wesley, Reading, Mass. (in press).
16. KOBRINSKII, N. E., AND TRAKHTENBROT, B. A. *Introduction to the Theory of Finite Automata* (trans.). North-Holland, Amsterdam, 1965.
17. KORFHAGE, R. *Logic and Algorithms*. Wiley, New York, 1960.
18. McCLUSKEY, E. J. *Introduction to the Theory of Switching Circuits*. McGraw-Hill, New York, 1965.
19. MILLER, RAYMOND E. *Switching Theory*, Vol. 1. Wiley, New York, 1965.
20. MINSKY, M. *Computation: Finite and Infinite Machines*. Prentice-Hall, Englewood Cliffs, N.J., 1967.
21. MOORE, E. F. (Ed.). *Sequential Machines: Selected Papers*. Addison-Wesley, Reading, Mass., 1964.
22. NELSON, R. J. *Introduction to Automata*. Wiley, New York, 1967.
23. PETER, R. *Rekursive Funktionen*. Akademisi Kiads, Budapest, 1951.
24. PRISTER, M. *Logical Design of Digital Computers*. Wiley, New York, 1958.
25. ROGERS, H. *Theory of Recursive Functions and Effective Computability*. McGraw-Hill, New York, 1967.
26. SHANNON, C. E., AND MCCARTHY, J. (Eds.). *Automata Studies*. Princeton U. Press, Princeton, N.J., 1956.
27. TRAKHTENDROT, B. A. *Algorithms and Automatic Computing Machines* (Trans.). Heath, Boston, Mass., 1963.

RECEIVED DECEMBER, 1967; REVISED APRIL, 1968

On Pointers versus Addresses

AMIR M. BEN-AMRAM

Tel-Aviv University, Tel-Aviv, Israel

AND

ZVI GALIL

*Tel-Aviv University, Tel-Aviv, Israel,
and Columbia University, New York*

LISP programmers know the value of everything, but the cost of nothing. — *Alan Perlis*

Abstract. What is the cost of random access to memory? This fundamental problem is addressed by studying the simulation of random addressing by a machine that lacks it, a “pointer machine.” The problem is formulated in the context of high-level computational models, allowing the use of a data type of our choice. A RAM program of time t and space s can be simulated in $O(t \log s)$ time using a tree. To enable a lower-bound proof, we formalize a notion of *incompressibility* for general data types. The main theorem states that for all incompressible data types an $\Omega(t \log s)$ lower bound holds.

Incompressibility trivially holds for strings, but is harder to prove for a powerful data type. Incompressibility is proved for the real numbers with a set of primitives that includes all functions that are continuous except on a countable closed set. This may be the richest set of operations considered in a lower-bound proof.

It is also shown that the integers with arithmetic $+$, $-$, \times and $\lfloor x/2 \rfloor$, any Boolean operations, and left shift are incompressible. The situation is reversed once right shift is allowed.

Categories and Subject Descriptors: E.1 [Data Structures]; F.1.1 [Models of Computation]; F.2.0 [Analysis of Algorithms and Problem Complexity]; General

General Terms: Algorithms, Theory

Additional Key Words and Phrases: Incompressibility, random access memory, pointer structures

1. Introduction

What should be the capabilities of an abstract “computer” for theoretical study? This question is of fundamental importance in complexity theory. The quest for bounds on the complexity of problems calls for a machine model that is both realistic and theoretically accessible. The random access machine (RAM) [3] seems to be the machine model in widest use. The general notion is that we want the RAM to model our experience in conventional programming. Thus, we let it handle “items” that are of a data type suitable for our needs:

The research of Z. Galil was supported in part by National Science Foundation grants DCR 85-11713 and CCR 86-05353.

Authors' addresses: A. M. Ben-Amram, Tel-Aviv University, Tel-Aviv, Israel; Z. Galil, Computer Science Department, Columbia University, New York, NY 10027.

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific permission.
© 1992 ACM 0004-5411/92/0700-0617 \$01.50

integers, real numbers, etc. We also let it use *indirect addressing*, which is the feature to which the RAM actually owes its name: the ability to access an arbitrary memory location, selected by an integer *address*. This corresponds to using arrays in a conventional programming language.

The selection of the data type that the machine should use—its data items and the primitive functions that may be applied to them—is an important decision, and since the first use of the RAM for complexity analysis, authors have considered this question, attempting to justify or evaluate possible choices, the data usually being integers [13, 17, 29], sometimes of limited size [15], or real numbers [9, 30, 36].

None of these authors considered the exclusion of the basic capability of indirect addressing. However, including it in our instruction set is a very important decision, since the use of *indirect addressing* in problems whose input and output are unrestricted data, which can be used—directly or indirectly—for addressing, makes the RAM very powerful (and hard to analyze). It allows for striking algorithms, such as distribution sort and table lookup by hashing [20].

Only in 1977 did Tarjan suggest the exclusion of this capability from our model in order to obtain nonlinear lower bounds for a data structure problem [38]. His machine model was first called “the reference machine” and later got the more intuitive name “pointer machine” [39]. This name suggests its replacement for RAM-type indirect addressing; it accesses memory locations by means of *pointers*. As in high-level languages such as LISP and Pascal, pointers are not numbers and are not subject to the same instructions.

Tarjan did not consider the power of pointer manipulation in general nor the effect of this change on the RAM, but rather used this machine as a context for developing a lower bound for a specific problem, that of maintaining disjoint sets on-line. To this end, he considered mainly the structures held in the machine's memory, while forcing the machine to operate on its data symbolically. However, in realistic algorithms, computation with the input values is frequently of value.

A natural framework for presentation of algorithms is often derived from an actual programming language. We have already noted that the indirect addressing feature appears in most high-level languages, and thus forms a part of the framework usually assumed. We also noted that the *pointer* model is quite common, and in one language, LISP 1.x [37], this is the only method of memory access, which means that users of this language actually view the underlying machine model not as the common RAM but as a different machine, which we shall formalize and call the *LISP machine*. Actually, many natural, well-known algorithms are designed in the framework of pointer structures.

Our commitment to realistic algorithms motivates us to consider various data types, since the algorithm designer tends to use a data type convenient for his/her purpose; that is, we may assume the basic data items to be integer, real, etc., and the choice of primitive operations may also be varied. In contrast with common practice, we do not define our models with a specific data type; our aim is to cope with this variety. In fact, we wish to formulate concrete definitions of generalized computational models so that the liberty, which one usually takes in describing algorithms, will be reduced to specifying a model in our class. This may also allow such algorithms to compare with compatible lower bounds. The basic model consists of a “CPU”—a device that executes a

program and has access to input and output tapes, a finite set of registers, and a "memory unit" that holds information for use by the CPU. The memory model makes the difference between a RAM and a LISP machine, while their CPU uses the same data type that may be varied. Our main results are tight bounds on the degradation in performance suffered by the RAM in its transformation to a LISP machine. The bounds are clearly affected by the choice of the data type.

A data type \mathcal{T} is defined by the *domain* of data values and the *primitive function set* that specifies the basic operations that the machine can apply to its data. We distinguish the *primitive set* of a data type from the *instruction set* of a machine; a machine model such as the RAM may be associated with various data types, and a data type may be put to use in various models. Symbolically, $\mathcal{T} = (\mathcal{D}, \mathcal{F}, <)$, where \mathcal{D} is the domain, \mathcal{F} the primitive function set, and $<$ an order predicate on \mathcal{D} . The machines considered in this paper are of the "high-level" breed: they can add, subtract, and compare numbers and input or output an arbitrary value. Since we are free to choose the data type, much power can be encompassed in the primitive operations on the data. This is an insurmountable barrier to lower-bound proofs. We overcome this problem by defining a class of *incompressible* data types. The *incompressibility* property is inherent in the data type; it is not related to a particular machine model and therefore is a general tool for deriving lower bounds, not only for the LISP machine. By analyzing several data types with respect to incompressibility, we justify the assumption of this property in propositions pertaining to a wide range of models; moreover, incompressibility suggests itself as a criterion for the plausibility of a data type.

The models are defined in Section 2. In Section 3, we give a strict definition to the problem of simulating a \mathcal{T} -RAM by another \mathcal{T} -machine. We consider the general problem of simulating a RAM program (with respect to its input and output). To handle this general problem, we define a specific problem for solution by the simulator. We show that the complexity of this solution determines the most general bounds on complexity of an on-line simulation of the RAM. This specific problem is called RASUS—random access storage unit simulation. It requires the simulator to act as a storage unit accepting LOAD and STORE commands. Its task is to answer a LOAD command by the value last stored in the given address.

On LISP machines, this task is easily accomplished using a binary search tree. The stored data are added to the tree with their address as key and retrieved in the usual fashion. By using balanced trees we get our upper bound, $O(t \log s)$.

This upper bound seems bound to be optimal. How can we store n items in a linked structure of out-degree 2, without having paths of length at least $\log n$? Indeed, the upper bound is optimal if our "items" have an atomic nature, as do symbols [39]. However, once we are allowed to operate on our data with nonelementary functions, that is, *compute* with them, it is no longer true that each input value must be stored separately. The machine may encode the input in its memory in any way that allows it to decode the information it may need. An example of how a powerful data type enables a faster simulation is given by the data type of integers with *shift* (Section 7), where we have a simulation in $O(t\alpha(s))$ time where α is a (very slow increasing) functional inverse of Ackermann's function. Indeed, this speed is gained by encoding the data in a compact structure. This seems as an unfair trick. In the area of low-level

models, such as Turing machines, lower bounds have been based on the knowledge that a string of length n may require n storage cells for representation [28]. Our new notion of *incompressibility* characterizes high-level data types in this respect. The precise definition, in order to characterize the data type and not a particular machine, is given in terms of decision trees that compute functions on \mathcal{Q}^n . The definition states that finite decision trees over an *incompressible* data type cannot encode \mathcal{Q}^n in \mathcal{Q}^m for $m < n$.

Our main theorem states that for such a data type, the machine cannot encode its input in a small memory graph, which entails the desired lower bound. The proof shows that encoding and decoding a memory graph is equivalent to transforming data tuples, as in the definition of incompressibility.

Finite data types provide a simple example of incompressibility, which is (implicitly) the fact used in the theory of Kolmogorov complexity [22, 28]. The power of the new definition is in its applicability to more sophisticated data types, and we proceed to show that useful data types are incompressible. The first data type we deal with is the real numbers. For this data type, we define a class of primitive functions that is larger than any practical set, and obviously stronger than the sets assumed in some older lower-bound proofs [6, 8, 30, 36], as well as other attempts to characterize real computable functions [9]. An important representative is the well-known *floor* function.

Denote by \mathcal{F} the set of functions $f: \mathbb{R}^k \rightarrow \mathbb{R}^m$, for $k, m > 0$, such that for some countable, closed set $C \subset \mathbb{R}^k$, f is continuous in $\mathbb{R}^k - C$. We show that the data type of real numbers where all primitive functions are in \mathcal{F} is incompressible. The proof makes use of the topological invariance of dimension, together with Baire's category theorem.

The last result is extended to the field of rationals as well. This relates to previous work [3, 8] in the same fashion as our result about the reals.

We then proceed to the most important of all, the integer data type. As mentioned above, we show that with a suitable instruction set, the data type is compressible. However, once we exclude *right shifting*, we get an incompressible data type, allowing (besides the standard operations) multiplication, all Boolean (bitwise) operations, unconstrained left shift and integer division by 2 (constrained right shift), and even exponentiation. Actually, we add all the operations assumed in the references, as long as the right shift is excluded [13, 17, 29]. The proof of incompressibility draws from the intuition that all these operations cannot retrieve information about the lower bits of a number that is "buried" in higher bits.

We conclude with historical notes on previous study of pointer machines.

A preliminary version of some of the results appeared in the first author's M.Sc. dissertation [5] and as an abstract in [6].

2. Definition of the Models

A *high-level model of computation* (or *machine*) consists of a read-only input tape, a write-only output tape, a finite number of registers, and (usually) a memory. Each of the input and output tape squares, as well as the memory cells, contains one item of the *ground data type* or *domain* of the machine. A random access machine is a high-level model where the memory cells are addressed by nonnegative integers.¹ It is assumed that the intersection of the

¹ \mathbb{N} denotes the set of nonnegative integers.

domain and \mathcal{A}' is a prefix of the latter, possibly infinite. We shall denote the number of registers by N_{reg} and the registers themselves by $R_1, R_2, \dots, R_{N_{\text{reg}}}$. A well-known fact about the RAM is that the actual number of registers is not important for our analysis, since memory cells can be used instead of missing registers, provided we have a certain (fixed) minimum number of registers needed to execute the necessary instructions. This replacement will cost us only a constant factor in running time. We make it a general requirement of high-level models that this property hold. Thus, in describing any single program for a high-level machine, we shall impose no limit on the number of registers used, but only demand that their number be bounded (which is always the case for a fixed program).

The program for the machine is not stored in memory. Thus, we are assuming the program does not modify itself. It consists of a (finite) sequence of (optionally) labeled instructions. The set of instructions consists of

- | | | |
|--|--------------|--------------------------|
| (1) <i>I/O instructions:</i> | <i>READ</i> | R_i |
| | <i>WRITE</i> | R_i |
| (2) <i>Standard register operations:</i> | <i>SET</i> | $R_i, \text{constant}$ |
| | <i>SET</i> | R_i, R_j |
| (3) <i>Flow control instructions:</i> | <i>JUMP</i> | <i>label</i> |
| | <i>JEQL</i> | R_i, R_j, label |
| | <i>HALT</i> | |
| (4) <i>Memory access instructions:</i> | | |

For the RAM, the last group includes the familiar

<i>LOAD</i>	R_i, R_j
<i>STORE</i>	R_i, R_j

which access the memory cell whose address is in R_j . This is *indirect addressing*. To access a cell of constant address, we can SET a register to the address. Thus, we may dispense with direct load and store. The address must belong to \mathcal{A}' ; otherwise, the program is invalid and will not be considered. We use the notation (a) for the contents of the cell addressed by a .

A fifth group of instructions is the *type-specific instructions*, also called *arithmetic* (although they may not be related to conventional arithmetic operations). This is a set of instructions that operate on registers only and implement functions on the machine's *domain*. All these instructions have the form

$$R_i \leftarrow f(R_{i_1}, \dots, R_{i_k}), \quad 0 < i, i_j \leq N_{\text{reg}} \quad \text{and} \quad f \in \mathcal{F}$$

where \mathcal{F} is the *primitive function set* of the machine. An additional domain-dependent instruction is

JGTR $R_i, R_j, \text{label},$

which compares data elements and branches to *label* if the value in the register R_i is greater than that in R_j . We do not impose any restrictions on the "greater than" relation except that the integers included in the domain are supposed to retain their natural order.

The ordered triple $\mathcal{T} = (\mathcal{G}, \mathcal{F}, <)$ of the machine's domain, primitive function set, and order relation is called the *data type* of the machine. A machine of type \mathcal{T} is called a \mathcal{T} -machine. A *parametric* family of machines is defined by a family of types $\mathcal{T}_i = (\mathcal{G}_i, \mathcal{F}_i, <)$, where the set of instructions is fixed though their semantics are necessarily dependent on the domain. This allows us to write one program that will run on the entire family.

The semantics of a program need not be explained in detail. It is assumed that the machine starts up with all registers zeroed and a blank memory (which, for a RAM, is defined to be all zeros). The machine then advances, according to program instructions and input data, between instantaneous states described by the contents of the registers, the program counter, and the contents of nonblank memory (which is always finite).

We are now ready to define the *LISP machine* (LM). This is a high-level machine whose *storage model* is a directed graph. To be specific, its memory is modeled as a finite set $\mathcal{S} \subset \mathcal{D} \cup \mathcal{P}$, \mathcal{P} being the set of *dotted pairs*, which is an infinite set satisfying $\mathcal{P} \cap \mathcal{D} = \emptyset$. This set is augmented with two operators $CAR, CDR: \mathcal{P} \rightarrow \mathcal{S}$, associating to each *dotted pair* two "pointers" to other "memory locations." We select a special constant $nil \in \mathcal{D}$ and define these functions as nil for arguments not in \mathcal{P} . In accordance with LISP conventions, we refer to the elements of \mathcal{D} as *atoms*.

To simplify matters, we regard the registers too as containing *pointers*. Technically, we have a "register contents" function $REG: \{1, 2, \dots, N_{reg}\} \rightarrow \mathcal{S}$ so that $REG(i)$ is the "contents" of register i . The number of registers is still considered to be an unspecified constant. It is not hard to show that, ignoring a constant factor change in running time, any bounded number of registers (above a necessary minimum) suffices for any program. The state of an LM is given, as for the RAM, by the contents of all the registers, the program counter, and the nonblank portion of memory. "Nonblank memory" means here the set of cells accessible from the registers via chains of CAR s and CDR s of any length. During the execution of the program, this set may grow as new dotted pairs are created, or shrink—when all references to a cell disappear. Since a cell that is "unlinked" from the structure can no longer influence the behavior of the machine, it can be assumed that such dotted pairs are deleted from \mathcal{S} .

The *memory image* of an LM (in some state of execution) can thus be said to have a *structure*, which is a directed graph whose nodes are $\{R_1, R_2, \dots, R_{N_{reg}}\} \cup (\mathcal{S} \cap \mathcal{P})$, and arcs naturally defined by the functions REG, CAR , and CDR , restricted to the set of nodes.

This structure is independent of the *data type* of the machine; the data type gets into the picture when we consider the "arcs" leading outside the structure, that is, the values of the REG, CAR , and CDR functions which are in \mathcal{D} . Given the structure alone, we can list the "names" of the missing arcs; Figure 1 explains this concept (Note: When drawing structures, the CAR of a dotted pair will be drawn by convention as an ordinary arrow, and the CDR as a double arrow).

This list is fixed by the memory structure and is independent of any data values; complementing it with a matching list of data values completes the description of the LM 's memory.

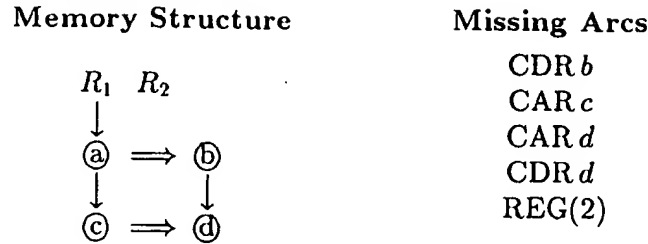


FIGURE 1

In summary, the instantaneous state of a LISP machine is defined by its “program counter” and the current *memory image*, described as the pair (*structure, content*) where *structure* is the memory graph and *content* is a list of domain values, the length of which is fixed for any specific *structure*.

We define the *height* of a memory image to be one plus the height of a BFS tree of its *structure* (the addition of one counts for the “missing arc” at the end of the path).

The instruction set of the \mathcal{F} -LM is the same as that of the \mathcal{F} -RAM, except for the memory access instructions that are replaced by a set of instructions for operating on dotted pairs. A valid program does not apply any of the arithmetic instructions to a dotted pair. We consider only valid programs.

Let us describe now the new instructions of the LISP machine:

$CONS \quad R_i, R_j, R_k.$

This instruction adds to \mathcal{S} a *new* dotted pair P whose CAR and CDR are $REG(j)$ and $REG(k)$, respectively, and sets $REG(i) = P$.

$LCAR \quad R_i, R_j$

This instruction sets $REG(i)$ to $CAR(REG(j))$.

$LCDR \quad R_i, R_j$

This instruction sets $REG(i)$ to $CDR(REG(j))$.

The instruction JEQL is also extended to test for equality of pointers. Thus it branches if $REG(i) = REG(j)$ regardless of whether these values belong in \mathcal{D} or in \mathcal{P} .

Another useful instruction would be a type test instruction, that is, something like “jump if $REG(i)$ is a dotted pair,” allowing a program to avoid applying pointer operations to data values and vice versa. It is easily seen, however, that programs using this instruction can be rewritten without it with at most a constant factor loss in running time; so the inclusion or omission of this instruction is not important.

The next two instructions separate the world of LISP machines in two, following an important distinction in the LISP language. In the world of *pure* LISP, the above instructions are all there is. The following two instructions distinguish **standard (full)** LISP from **pure** LISP, in that they add the capability to *change the contents* of an existing memory location, where in **pure** LISP a

memory location does never change its value. The instructions are:

$RPLACA \quad R_i, R_j$

Change $CAR(REG(i))$ to $REG(j)$.

$RPLACD \quad R_i, R_j$

Change $CDR(REG(i))$ to $REG(j)$.

If applied to a domain value, the effect of these instructions is undefined.

To sum up, we have in fact two distinct LISP machines: the pure LISP machine, PLM, and the full LISP machine, FLM. The distinction between the two highlights a theoretical problem: what are the implications of including or excluding the capability of changing part of a dotted pair (or more generally a record) in situ. The outstanding effect of this capability is that cycles can appear in the memory structure; the structure of a PLM's memory is always an acyclic graph, topologically ordered by the time the nodes were created. Thus it would be interesting to find the complexity of simulating the FLM on the PLM. For the RAM simulation problem, however, the machines exhibit similar performance; that is, none of our results separates the two. The common name "LISP machine" will be used in a generic sense; that is, a statement mentioning LM is to be true when either PLM and FLM are consistently substituted for LM.

Each LISP machine starts off with all the registers initialized to *nil*, and then changes its state according to the program instructions and input data.

To define the running time, or *time complexity*, of a RAM or LM program, we shall use the *uniform cost model*; that is, we charge one "time unit" for each instruction performed. Uniform cost is also applied to space complexity in the RAM, which is defined by the number of memory locations used.

Programs for high-level models will be written in *pidgin algol* with simple variables and control structures and also procedure calls. It is well known how to compile this language into "machine code" and evaluate their running time (e.g., [3]). It is helpful to note that stacks, a useful data structure, are easily implemented in pure LISP. Stacks are represented by the standard LISP data structure known as *list*: For completeness, here is its definition.

A list is either *nil*,
or it is a dotted pair, whose CDR is a list.

The CARs of the dotted pairs comprising the list are called the *list elements*.

3. The Simulation Problem

3.1. SIMULATION OF THE STORAGE UNIT. The only difference between high-level models of the same data type is the memory model. Thus, when asking to compare the computational power of two machines, it is natural to see the question as that of comparing the power of their *storage units*.

The idea of the *storage unit* of a RAM follows naturally from the definition of the model. It is supposed to be some kind of machine, which can be given *commands* of the form *STORE d, a* and *LOAD a* where $d \in \mathcal{D}$ and a (the *address*) is in $\mathcal{S} \cap \mathcal{A}$. The machine should output in response to each *LOAD* the value last stored into the specified address by a *STORE* or 0, otherwise.

A *simulation of the RAM storage unit* (RASUS, for short) consists of two programs for a high-level model. One program is used to carry out a *STORE*:

and the other to carry out a LOAD. Initially, the machine starts with a fixed memory structure that corresponds to an all-zero memory of the RAM. Each STORE is carried out by placing a and d in designated registers and executing the store program. Each LOAD is executed by placing a in its designated register and executing the load program, whose task is to deposit in the d register the value last "stored" into address a . The memory image is not changed in any way between execution of the programs. Instead of providing the initial memory structure, it may be convenient to have an *initialization program* in addition to the previous two. This is a one-shot program with no input, so its running time is fixed and of no interest.

Let $T_{sim}(u)$ be the total time spent by the machine in performing the simulation for the sequence of commands u . Define the *RAM time* $t(u)$ to be the number of instructions in u , that is, its length; and define the *RAM space* $s(u)$ to be the number of distinct addresses that appear in STORE commands in u . Then, we measure the *time complexity* of the simulation by the function

$$T(t, s) = \sup_{\substack{u: t(u) \leq t \\ s(u) \leq s}} T_{sim}(u).$$

We can also consider $T_{load}(t, s)$ and $T_{store}(t, s)$, the worst-case time needed by the *load* and *store* programs, respectively, after any preceding command sequence u such that $t(u) \leq t$ and $s(u) \leq s$.

If $T_{load}(t, s)$ and $T_{store}(t, s)$ are bounded by the function $f(t, s)$, we say the simulation has a *delay* of $f(t, s)$. We further attribute it as *uniform delay* if

$$f(t, s) = O\left(\frac{T(t, s)}{t}\right).$$

A simulation having constant uniform delay is said to be *real time*.

3.2. MACHINE SIMULATION. A fundamental approach to the comparison of random access and LISP machines is to look for bounds on the gap between the performance of *any program* on the RAM and a program for the same problem on the LM. Excluding the case of a RAM with finite memory, simulating an LM by the RAM is both simple and efficient (real time). Obtaining the complexity of simulating RAM programs by the LM is less trivial, and we shall see that we can learn of time bounds for such simulations by studying the complexity of RASUS on the LM. In fact, we show that for any high-level model, the complexity of RASUS determines the time bounds on simulating the RAM in general.

We define one program as simulating another one if their external behavior matches. By external behavior we mean the sequence of input instructions performed, output produced (for a given input), and halt (if done) throughout the execution of the program. Note that this is necessarily an *on-line* simulation (the alternation of input and output operations is preserved). Let $t_0 < t_1 < \dots < t_i < \dots$ be the time steps at which the simulated program makes an external operation. Let $t'_0 < t'_1 < \dots < t'_i < \dots$ be the corresponding time steps of the simulation. Let s_i be the space used by the RAM in the time interval $[0, t_i]$. We say that the simulation time is governed by a function $T(t, s)$ if for all input sequences, for all i such that $a < t_i \leq t$ and $s_i \leq s$, we have

$$t'_i \leq T(t, s).$$

The simulation is said to be *real time* when there exists a constant $c > 0$ such that for all input sequences we have

$$t'_i - t'_{i-1} \leq c(t_i - t_{i-1}) \quad \forall i \geq 0.$$

In general, we say that the simulation has a *delay* governed by $f(t, s)$ if for all input sequences we have

$$t'_i - t'_{i-1} \leq f(t_i, s_{t_i}) \cdot (t_i - t_{i-1}) \quad \forall i \geq 0.$$

The simulation *achieves time* $T(t, s)$ *with uniform delay* if its running time is governed by $T(t, s)$ and its delay by

$$O\left(\frac{T(t, s)}{t}\right).$$

3.3. EQUIVALENCE OF THE PROBLEMS. This subsection is devoted to proving that the general problem of simulating the RAM is equivalent to the specific problem of random access storage unit simulation (RASUS). In the proof, we need the following technical lemma:

LEMMA 1. *If RASUS can be solved by a high-level machine in time $T_1(t, s)$, then for any $k > 0$ there is a RASUS solution for the same machine whose running time T_2 satisfies*

$$T_2(kt, s) \leq c_k T_1(t, s),$$

where c_k is a constant.

PROOF. Let RASUS_1 be the RASUS solution that runs in time T_1 . We obtain the desired solution RASUS_2 by duplicating the register set used by RASUS_1 $2k^2$ times, as well as the code for the procedures, so that we actually have $2k^2$ RASUS_1 clones operating concurrently, to which we refer as $D(i, j)$ (for *data*) and $T(i, j)$ (for *timestamp* where $0 \leq i, j < k$). In addition, we count the STORE commands in N_{store} . Here is the code for RASUS_2 , under the simplifying assumption that all the duplicates share the registers a and d that are not modified except for depositing the result of a successful LOAD. Therefore, these parameters need not be passed explicitly. The *load* procedure uses two auxiliary registers, r and t .

```

init2:
  Nstore, Nload ← 0
  for 0 ≤ i, j < k
    init1(D(i, j))
    init1(T(i, j))
  end for
store2:
  Nstore ← Nstore + 1
  i ← Nstore mod k
  for j = 0, ..., k - 1
    store1(D(i, j))
    d ← Nstore
    store1(T(i, j))
  end for

```

```

load2:
  Nload ← Nload + 1
  j ← Nload mod k
  r ← 0
  t ← 0
  for i = Nstore + 1, ..., Nstore + k (mod k)
    load1(T(i, j))
    if d > t then
      t ← d
      load1(D(i, j))
      r ← d
    end if
  end for
  d ← r
end for

```

The correctness of the solutions stems from the fact that every *store* affects each column of the $k \times k$ matrix, *load* scans a whole column, and the timestamps ensure that the last STORE into the sought address determines the output.

Thus, every *store* and *load* calls at most $2k$ times the corresponding procedure of RASUS₁, while each of the RASUS₁ duplicates only receives up to $1/k$ of the commands. This gives immediately the desired time bound. The constant c_k seems to be $O(k)$, but we have multiplied the number of registers by k^2 ; so we may want to take into account the cost of absorbing this growth in number of registers. On the LISP machines, this problem may be solved with a bit of programming by linking the memory images of the k^2 RASUS clones in a suitable list structure. Then, the code need not be duplicated, and c_k is a genuine $O(k)$. \square

LEMMA 2. *The general simulation of the RAM and the RASUS problem are real-time equivalent on any high-level machine. That is, if the machine can solve RASUS with time complexity $T(t, s)$, then it can simulate every RAM program in time governed by $3T(t, s)$. Vice versa, if there exists a function $T(t, s)$ such that every RAM program can be simulated by our model in time governed by $T(t, s)$, then there it can solve RASUS in time complexity $T_{\text{RASUS}}(t, s) \leq cT(t, s)$ for some constant c . This relationship holds also for the delay of uniform delay solutions.*

PROOF

(\Rightarrow) Assume that there is a solution to the RASUS problem that runs in time $T(t, s)$. Let an arbitrary RAM program be given. All the instructions appearing in the program (except for the memory access instructions) can be directly executed by the simulator (they appear in its instruction set). Thus, only the STORE and LOAD have to be replaced to form the simulation. These two will be replaced by invocations of the *store* and *load* program of the RASUS: to avoid any conflicts we may equip them with a separate set of registers. Thus, the execution of a memory access instruction will be replaced with moving the operands to or from the dedicated registers (at the cost of two instructions) and executing the RASUS procedure.

Assume that for some input, the RAM program uses space s while performing the first t instructions. Let t_1 be the number of STORE and LOAD

instructions performed. Our simulation will run in time bounded by $t - t_1 + (2t_1 + T(t_1, s)) = t + t_1 + T(t_1, s)$. Since necessarily $T(t, s) \geq t$, and $T(t, s)$ is monotone increasing in t , we have

$$t + t_1 + T(t_1, s) \leq 3T(t, s).$$

Using the definition of the simulation time complexity in Subsection 2.2, we obtain that our simulation has time complexity bounded by $3T(t, s)$.

This argument can be readily extended to the delay of uniform delay solutions, which completes the proof.

(\Leftarrow) Assume that the \mathcal{T} -machine can simulate the RAM in time $T(t, s)$. We shall build a solution to the RASUS problem running in similar time. Such a solution must be a pair of programs as specified in the RASUS definition (Subsection 2.1). That subsection also defines the time complexity of RASUS, a function of the *RAM time* and *RAM space* of the input sequence. However, the bound that we are given is on the time that simulation of a RAM *program* may require on the \mathcal{T} -machine; and we must convert it to a bound on the RASUS time, dependent only on the aforementioned parameters. The conversion is achieved by choosing a RAM program that is an "interpreter": it accepts the command sequence as input and executes it. Its time and space complexity will be seen to be proportional to the *RAM time* and *RAM space* of the input sequence, allowing us to transform a simulation of this program to a RASUS solution of similar complexity.

The RAM program that we use is shown in Figure 2.

This program executes an infinite loop. In each iteration of the loop, it reads a command code (one of two predefined constants) and performs the operation requested, reading the operands for this command and for a LOAD, writing out the result. Denote by $\tilde{t}(u)$ and $\tilde{s}(u)$ the time and space, respectively, required by this program for handling a command sequence u . It is easy to see that

$$\begin{aligned} \tilde{t}(u) &\leq c_1 t(u) \quad \text{for some integer } c_1 \text{ and} \\ \tilde{s}(u) &= s(u) \end{aligned} \tag{1}$$

where $t(u)$ and $s(u)$ are defined as in Subsection 3.1. Let *prog* be a program for the given machine that simulates the above program in time $T(\tilde{t}(u), \tilde{s}(u))$. From *prog*, we form a RASUS solution, namely, the three procedures *init*, *store*, and *load*. The difficulty in obtaining these procedures lies in the fact that our program may be more complicated than a "read and execute" loop (which would readily yield a RASUS). There may be several READ commands interspersed throughout the program, which alternately assume the role of reading the command code or operands. Moreover, this program runs forever. Our procedures are one-shot programs (although the memory image is preserved between the calls), and therefore, to simulate the actions of this program, its "program counter" has to be restored as well. Since the program is fixed, we can label the points where we may stop with a finite set of labels and code a multi-way branch which, according to a preserved index, continues execution at the desired point.

Each of the procedures will follow *prog* up to a certain point: The initialization procedure consists of the (unique) sequence of instructions executed by *prog* up to the first READ instruction. The *store* procedure replaces reading the command code by setting it to STORE and reading the two operands by


```

repeat forever
  read command
  if command = STORE then
    read  $d, a$ 
     $(a) \leftarrow d$ 
  else /* /* command = LOAD
    read  $a$ 
    write  $(a)$ 
  end if

```

FIGURE 2

getting them from the designated registers. At the next READ, it stops. The *load* procedure is similar. We are now ready for coding the solution, where we assume the arguments to the procedures to be passed in registers R_a and R_d and make use of three additional registers, R_{pc} (for saving the program counter), R_{inp} (for simulating input operations), and R_{inps} (for counting input operations).

The RASUS procedures start with the following lines:

```

init:
   $R_{inps} \leftarrow 0$ ; no operands to "read"
  JUMP prog
store:
   $R_{inp} \leftarrow STORE$ 
   $R_{inps} \leftarrow 2$ ; "read" both  $d$  and  $a$ 
  JUMP  $L_{R_{pc}}$ ; a multi-way branch
load:
   $R_{inp} \leftarrow LOAD$ 
   $R_{inps} \leftarrow 1$ ; "read"  $a$ 
  JUMP  $L_{R_{pc}}$ ; a multi-way branch

```

In addition, we substitute the following code segment for each occurrence of "READ R_i " in *prog*. Here k is a different integer for every READ instruction replaced, yielding a unique label L_k .

```

if  $R_{inps} = 0$  then
   $R_{pc} \leftarrow k$ 
  halt
else
  if  $R_{inps} = 2$  then
     $R_{inp} \leftarrow R_d$ 
  else
     $R_{inp} \leftarrow R_a$ 
  end if
   $R_{inps} \leftarrow R_{inps} - 1$ 
end if
 $L_k: R_i \leftarrow R_{inp}$ 

```

Each "WRITE R_i " is replaced by

$$R_d \leftarrow R_i.$$

Clearly, the running time of the RASUS obtained is linear in the running time of *prog*. That is, there exists an integer c_2 such that for every sequence u of

STORE and LOAD commands we have

$$T_{\text{RASUS}}(u) \leq c_2 T(\tilde{t}(u), \tilde{s}(u)) \leq c_2 T(c_1 t(u), s(u))$$

(without loss of generality, we may assume T to be increasing). We thus obtained a RASUS solution in time $T_{\text{RASUS}}(t, s) \leq c_2 T(c_1 t, s)$. The proof is completed by an application of Lemma 1.

This argument can be readily extended to show that if *prog* simulates the RAM with uniform delay, the RASUS procedures have a proportional uniform delay. \square

4. Time Bounds on the Simulation

Now that the tools are prepared, we set out to discover the complexity of simulating random access memory (or machines) by an LM.

4.1. AN UPPER BOUND

ALGORITHM 1. A search tree solution to RASUS.

This algorithm will not be specified in detail, since any of the well-known balanced-tree schemes will serve (see, e.g., [3]). These tree structures are used to store and retrieve data values by their *keys*, which are in our application the RAM addresses. The *store* program implements the *insert* operation on the tree, and the *load* program—the *find*. The tree structure can be very easily constructed of LISP cells, and the programs for *insert* and *find* can be readily written in the FLM language.

On the PLM, implementation of the tree algorithms may have to be changed, as they usually use the operation of changing the value at a node or the identity of its child. However, coercing them to the pure LISP model is not difficult. In all these schemes, the *insert* and *find* programs have the general form of a descent down a particular path from the root to a leaf, then an ascent back, during which modifications are made to the nodes on this path. The locality and order of changes allow for an easy implementation on the PLM, in which pointers to the nodes *neighboring* the path are saved in a stack while descending, and on the ascent, the path is rebuilt anew (with the modifications required); and the stack is used to connect it to the neighborhood. This is the “path-copying” technique mentioned in [31], where its feature of not modifying existing nodes is used to allow “past generations” of the tree to be retrieved.

The running time of this solution is clearly $O(t \log s)$, as $T_{\text{load}}(s) = T_{\text{store}}(s) = O(\log s)$ (by known properties of balanced trees). Thus, we get our upper bound:

THEOREM 1. *The LISP machine can simulate the RAM in $O(t \log s)$ time with uniform delay.*

4.2. A SIMPLE LOWER BOUND. The balanced-tree schemes have the reputation of optimality, and it is natural to conjecture that the FLM cannot do better than that in the RASUS problem. A “proof” of an $\Omega(t \log s)$ lower bound is easily obtained.

Consider the state of the FLM just before initiating the *load* program, sometime during a simulation. In the first instruction executed by the

program, the only *atoms* it can access are those currently linked to registers, that is, N_{reg} atoms at most. By performing another instruction, no new atoms in memory can be retrieved via a register pointing to an atom, while a pointer to a dotted pair can be used to load any one of its CAR and CDR; thus in two instructions, up to $2N_{\text{reg}}$ previously stored atoms can be accessed. Extending this argument by induction, we get that in k instructions starting at any given state, up to $2^{k-1}N_{\text{reg}}$ different atoms previously stored in memory can be retrieved.

For any $s > 0$, let u be a sequence of s STORE instructions with addresses $0, 1, \dots, s-1$ and different data. After executing u , the simulating machine must be ready to handle correctly any command of the form *LOAD a* where a is one of the addresses in u . According to the previous argument, at least half of the atoms previously stored in memory require $\lceil \log s - \log N_{\text{reg}} \rceil$ steps to be retrieved. Actually, they are that far from the registers in the memory graph. Thus, a load command can now be issued that will force the machine to “travel” a distance of $\Omega(\log s)$ in order to find the requested atom. This argument entails a lower bound on the time for retrieval that holds even with unbounded preprocessing time. Therefore, it remains valid at any later time, and we can create an unbounded sequence of worst-case LOAD commands, obtaining a lower bound of $\Omega(t \log g)$ on simulation time.

The weakness of this argument lies in the assumption that when processing a store, the machine actually stores the given datum in memory and has to fetch it from this location for a load operation. In fact, the LISP machine might encode the simulated RAM's memory contents in a very clever way, giving rise to a better simulation. This way the *data type* of the machines comes into the picture; only by relying on properties of the data type we may avoid such pitfalls. For a start, the above proof is valid if our machines may handle *symbols*.

The term, *symbol*, is defined in [38, 39] referring to an item on which no operations are permitted except testing for equality. This definition coincides well with the usage of LISP, in which *atoms* may well be symbolic, in addition to numeric.

THEOREM 2. *Let \mathcal{F} be a data type containing an infinite set of symbols. Then a \mathcal{F} -FLM needs $\Omega(t \log s)$ time to simulate a \mathcal{F} -RAM.*

PROOF. None of the instructions in the FLM instruction set can “generate” a symbol; the only instructions whose execution may result in a register pointing to a new symbol are SET, LCAR, and LCDR. Therefore, the above argument applies. \square

4.3. OTHER DATA TYPES. The proof of the lower bound in the previous subsection relied heavily on the presence of “symbols” in the *domain* of the machines considered. Though it was stated that this is a natural way of thinking in LISP, it is not at all natural to the RAM. So our “goal” now is to show that the $\Omega(t \log s)$ bound carries over to RAM models with more useful data types.

In the following subsections three main models of the RAM, that is, three *data types*, will be analyzed in this respect. The three models are:

- (i) The *real number* RAM, whose domain consists of the real numbers.

- (ii) The *integer* RAM, whose domain is the integers.
- (iii) The *finite word* RAM, whose domain is finite and will be considered as consisting of k -bit words, where k is a parameter, making it a model family (Section 2).

The main tool for the analysis will be a notion of *incompressibility*. This notion is the actual feature of the *symbol* data type that is required for the proof. In Section 5, this notion is defined, and a general proof of the lower bound using this notion is given. This feature is proved for case (iii) in Section 5 and for (i) in Section 6. Section 7 is devoted to the integer case; it will be seen that the set of primitive functions should be chosen with care if this data type is to be incompressible.

The wide success of the lower-bound proof can be interpreted as an assurance of the general validity of accessibility considerations. The fact that it fails for some data types may be interpreted as indicating that incorporating them in our model is far fetched.

5. Incompressibility and Lower Bounds

5.1. DEFINITIONS. The notion of an incompressible data type is that of a data type \mathcal{T} such that a \mathcal{T} -machine is not able to "encode" a number of data atoms in a smaller number and so economize on memory accesses. For a precise definition, we need a simple model of computation that allows us to concentrate on properties of the data type.

Let $\mathcal{T} = (\mathcal{D}, \mathcal{F}, <)$. Each function in \mathcal{F} has as domain the set \mathcal{D}^n for some $n > 0$; for simplicity of exhibition, we include also in the sequel functions on \mathcal{D}^0 , which represent constants. We denote by \mathcal{F}^* the *closure* of \mathcal{F} under the following operations. *Aggregation* combines some functions $f_1, \dots, f_n: \mathcal{D}^m \rightarrow \mathcal{D}$ to one vector-valued function $f = (f_1, \dots, f_n): \mathcal{D}^m \rightarrow \mathcal{D}^n$. *Composition* is used to apply a function to the result of another: composing $g: \mathcal{D}^n \rightarrow \mathcal{D}$ with $f: \mathcal{D}^k \rightarrow \mathcal{D}^n$ yields $g \circ f: \mathcal{D}^k \rightarrow \mathcal{D}$.

The model of computation we shall use is the \mathcal{T} *decision tree*. A decision tree of n inputs is an ordered finite binary tree each of whose internal nodes represents a "decision." Actually we associate with each internal node v a function $f_v \in \mathcal{F}^*: \mathcal{D}^n \rightarrow \mathcal{D}$. The computation begins in the root; in every internal node v reached, the function f_v is applied to the input values, and the computation now proceeds according to its value, "control" moving to the right child if it is greater than zero and otherwise to the left. The expression "greater than" relates to the order relation $<$ of the data type \mathcal{T} . Upon reaching a leaf, the computation ends with its result given by the leaf's function, applied to the inputs.

We remark that this is not a comparison tree; since we compare the outcome of functions, whose exact nature is not specified, any test that we wish to allow our models to perform may be included in the tree.

The terminal (leaf) functions have the range \mathcal{D}^m (for some fixed m). We say the tree computes a function from \mathcal{D}^n to \mathcal{D}^m .

It is quite easy to see that each function computable by a \mathcal{T} -PLM or a \mathcal{T} -FLM (as well as a \mathcal{T} -RAM) program can be computed by an \mathcal{T} decision tree (\mathcal{F} relating to \mathcal{T} as above), provided the program has a finite bound on its running time (we just "unroll" the program and make explicit all calculations to avoid the use of memory). If we are to compute a function of $\mathcal{D}^n \rightarrow \mathcal{D}^m$, the

running time of our program may depend on n and m , and we impose no restrictions on its rate of growth; the finiteness requirement disallows programs whose running time grows indefinitely for fixed n and m , depending on the input values (programs of *nonuniform* complexity). Such a program may "compress" any effectively countable domain, for example, by the conventional "pairing functions." Therefore, we shall use the term, *finitely computable*, to describe a function that can be computed with uniform complexity, where \mathcal{F} can be understood from context.

Definition. A data type $\mathcal{T} = (\mathcal{D}, \mathcal{F}, <)$ is called *incompressible* if for all $n > 0$, there is no pair f_n, g_n of functions, computable by \mathcal{F} decision trees, such that f_n maps \mathcal{D}^n into \mathcal{D}^{n-1} , and g_n is its inverse.

\mathcal{T} is incompressible if and only if for all $m < n$ there is no pair f, g of finitely computable functions such that f maps \mathcal{D}^n into \mathcal{D}^m , and g is its inverse.

Remark. The incompressibility of a data type of *finite domain* is implicit in the work of Kolmogorov [22]. The value of this feature to proofs in computational complexity was advocated by Paul et al. [27, 28] and other work. Our notion of incompressibility is more general in nature, but is used towards the same end.

5.2. THE LOWER BOUND. We now proceed to give the lower bound proof for incompressible data types. To this end, we have to consider LISP machines and their memory images instead of the general decision trees. So instead of programs computing functions of their inputs, we consider programs that *transform memory images*. We say a program transforms the memory image M_1 to the memory image M_2 whenever having set the LM's memory to M_1 and started the program, it halts after changing the memory image to M_2 . For sets of memory images \mathcal{M}_1 and \mathcal{M}_2 , we say that \mathcal{M}_1 can be encoded in \mathcal{M}_2 if there is a program that transforms each element of \mathcal{M}_1 to an element of \mathcal{M}_2 , and another program that "decodes" them correctly, that is, implements the inverse transformation. Both of these programs are required to have their running time bounded by a constant (generally dependent on the sets).

Say that a memory image is *rooted in R_1* if all the other registers are *nil* for that image.

Consider memory images that are lists (cf. Section 2), rooted in R_1 . It is easy to see that the incompressibility of \mathcal{T} , defined generally in the previous subsection, can be formulated for LMs as:

\mathcal{T} is incompressible if the set of lists of n atoms cannot be encoded in the set of m -atom lists, for any $m < n$.

The lower-bound proof will show that if an FLM can simulate the RAM "too fast," that is, it has RASUS programs of low running time, then we can obtain from these programs an encoding of n -element lists in m -element lists, for $m < n$. Actually the fast RASUS solution amounts to an encoding of n elements in a memory image which is not necessarily a list, and we must show that we can further encode these images in the set of short lists.

LEMMA 3. *The set of memory images that are rooted in R_1 and have height at most h can be encoded in the set of lists of length $\lfloor (3/4)2^h \rfloor$, provided that $\{0, \dots, 2^h - 1\} \subseteq \mathcal{D}$.*

PROOF. If $h = 1$, the lemma is trivial, since the image is already a list of one atom. We continue with $h > 1$.

Let M be a memory image, rooted in R_1 and having height at most h .

Recall the division of a memory image into *structure* and *contents* (Section 2). The contents of the memory image are actually *leaves* in a graph, whereas the structure is the subgraph excluding the leaves. Each leaf appears at the end of a directed path, and since M 's height is at most h , there are at most 2^{h-1} different paths to leaves. Hence, the number of leaves is at most 2^{h-1} .

The largest number of leaves occurs when M is a complete binary tree. Then it has 2^{h-2} nodes at level $h-1$, each one a parent of two leaves. If we had to encode such an image as a list, we would just have to list the values of the leaves; thus, lists of length 2^{h-1} encode these images.

Certainly, M is not necessarily a tree. To deal with general structures, we add to each atom in the list a *mark bit*, which is either 0 or 1. We encode the marks of pairs of consecutive elements in the list in the bits of one two-bit number (note that $\{0, \dots, 3\} \subseteq \{0, \dots, 2^h - 1\} \subseteq \mathcal{G}$). This defines the final form of our list: 2^{h-1} data atoms accompanied by 2^{h-2} mark atoms, encoding one mark bit for every atom. The length of the list remains fixed, so we may use array notation and write $A[0], \dots, A[2^{h-1} - 1]$ for the data atoms and $M[i]$ for the mark bit of $A[i]$ (actually these mark bits are paired as above).

To encode the memory image M in these two arrays, we use a standard correspondence between nodes in the BFS tree of the structure and binary numbers. This BFS tree is an ordered binary tree, with CAR preceding CDR for each dotted pair. We refer to the single dotted pair connected to R_1 as the root of the tree (and of the image), and neglect the arc leading into it (we deal only with arcs emanating from dotted pairs). Then, by the definition of height (Section 2) this tree's height is $h-2$.

The correspondence is obtained by assigning the value 0 to CAR and the value 1 to CDR. We denote by T the unique BFS tree that results from an ordered BFS search in which CAR is handled before CDR. Each node in the structure is the end of a unique path from the root of T , whose length is at most $h-2$. We label it by a number whose binary representation is $b_1 b_2 \dots b_{h-1}$, where b_i is the value of the i th arc along the path, if such an arc exists, and otherwise zero (b_{h-1} is necessarily zero, but it is convenient to keep it). Hence, the dotted pair linked directly to R_1 is labeled by zero, and the same for its CAR; its CDR is labeled 2^{h-2} . The range of possible labels is 0 to $2^{h-1} - 1$, corresponding to indices in A and M . Obviously, each number may be labelling more than one node. The following facts are easily established.

- (i) The set of nodes labeled by ν has the following form: u , CAR u , $\text{CAR}^2 u$, \dots , $\text{CAR}^l u$ for some $l \geq 0$. Of all these nodes, only the last one is a leaf in T (recall that a leaf of T is not necessarily a leaf of the structure).
- (ii) Let u be a node of label ν , and let k be its depth in the tree. Then, ν is a multiple of 2^{h-1-k} ; all the descendants of u in T have labels of the form $\nu + i$, $0 \leq i < 2^{h-1-k}$; and only descendants of u may bear labels of this form for $i > 0$.

We call a node $v \in M$ a *front node* if it is a dotted pair such that at least one of the arcs CAR v and CDR v does not belong to T . It is easy to see that

the memory image is reconstructible from the following information: (1) the labels of its front nodes, (2) the depths of these nodes in T , and (3) the CAR and CDR of each of these nodes. We show how to encode this information in the arrays A and M . Let v be a front node labeled ν , and k its depth in T . By Fact (ii), since $k \leq h - 2$, all the node labels must be even. If $k = h - 2$, then both $\text{CAR}(v)$ and $\text{CDR}(v)$ have to be atomic, and we set $M[\nu] = M[\nu + 1] = 0$ and $A[\nu] = \text{CAR}(v)$, $A[\nu + 1] = \text{CDR}(v)$. Otherwise, $k < h - 2$. Suppose that the CAR arc of v is not in T (CDR is handled similarly). Then by Fact (i) we know that ν labels no node of depth greater than k . Moreover, there may be only one front node with this label and CAR not in T . We set $M[\nu] = 1$ and $A[\nu] = k$. We save $\text{CAR}(v)$ in $A[\nu + 1]$, which is free since $\nu + 1$ might only label descendants of $\text{CAR}(v)$, had the CAR arc of v belonged to T (Fact (ii)). If $\text{CAR}(v)$ is atomic, we set $M[\nu + 1]$ to 0 and $A[\nu + 1]$ to the value of $\text{CAR}(v)$. Otherwise, we set $M[\nu + 1]$ to 1 and $A[\nu + 1]$ to the label of $\text{CAR}(v)$. Some of the array elements are not given meaningful values: These are the elements numbered $\nu + 2$ to $\nu + 2^{h-2-k} - 1$. However, this will not defy the correct decoding of the information, since, by scanning M sequentially, we identify these elements when processing ν and skip them afterwards. All other elements of M and A must have been set by the above procedure (which is an easy exercise to verify).

The above description, while not strictly formal, is not difficult to translate into an LM program, whose running time is bounded by a function of h alone. Likewise, it is easy to see that the construction can be reversed, and an inverse transformation can be programmed within a similar time bound. The difficult part of this program is the reconstruction of pointers to other nodes, which have not yet been created. This can be overcome by using two passes: in the first, we represent pointers to nodes by the nodes' number and a special mark. In the second pass, on the FLM we redirect the pointers to their intended target. On the PLM, we use a topological sort to obtain the sequence in which we have to create the nodes. Such a sequence will be obtained since the encoded image has been built by a PLM too. \square

THEOREM 3. *Let \mathcal{T} be an incompressible data type. Then a \mathcal{T} -FLM requires $\Omega(l \log s)$ time to simulate a \mathcal{T} -RAM.*

PROOF. Again, we consider the RASUS problem. We show that for all pairs $s > 0$, $n > 0$, there exists a sequence u of s stores and a sequence v of n loads, such that the simulation of uv must take time $> n \lceil \log s \rceil$. Assume to the contrary that there are a RASUS solution R and a pair $s > 0$, $n > 0$, such that for all sequences u of s stores and all sequences v of n loads, the simulation of uv takes at most $n \lceil \log s \rceil$ time. Without loss of generality, it may be assumed that the memory image left when any of the RASUS procedures terminates is rooted in R_1 . Fix this solution and the values of n and s for the rest of this proof. We show that it necessarily holds that after simulating u and some part of v , R is able to answer every load request using a memory structure whose height is bounded by $\lceil \log s \rceil$. We can effectively discover such a structure and obtain an encoding which contradicts incompressibility.

Let a list of s atoms be given. Make up a sequence u by storing these data into addresses $0, \dots, s - 1$. In the rest of this proof we deal with load commands from these addresses only. Call a load operation *costly* if it takes more than $\lceil \log s \rceil$ time units, and *cheap*, otherwise. A sequence v' of $k \geq 0$

load commands is defined as *costly* if when fed to R following u , each of its constituents is a costly load. By our assumption, there does not exist a costly sequence of length n . Hence, by pursuing any chain of costly loads, we must—after less than n operations—reach a state where every load operation is cheap. Therefore, we prune the memory image of R to the height of $\lfloor \log s \rfloor$, that is, change all pointers into farther dotted pairs to *nil*, the flow and outcome of every load operation will not be affected. Therefore, having found such an image, and pruned it, we have a memory image of height at most $\lfloor \log s \rfloor$ which encodes the s input elements.

To complete the proof, here is a description of the encoding and decoding programs. Note that their running time is bounded.

Encoding program:

Input: A list of s elements. L .

Output: A memory image of height $\leq \lfloor \log s \rfloor$.

Method: A dedicated set of registers is used for simulating R , whose procedures, $store_R$ and $load_R$, are only modified by adding an instruction counter t to $load_R$ that allows us to monitor its time. The procedure *CopyImage* copies the memory image rooted in one register to another one (a simple task on an FLM). On a PLM, we use a topological sort to obtain the order in which we create the nodes so that the pointers can be reconstructed. Such an order always exists, because the copied image was built by a PLM. The rest of the description lies in Figure 3, omitting the act of pruning, which is similar to copying the image.

Decoding program:

Input: A memory image produced by the encoding program.

Output: The reconstructed list.

Method: For $i = s - 1, \dots, 0$, copy the memory image to R_i and call $load_R(i)$. Push the retrieved value onto the list.

The conclusion of this construction is that we can encode the lists of s atoms in memory images at most $\lfloor \log s \rfloor$ high, rooted in R_i . By Lemma 3, these images can be further encoded as lists of $\lfloor (3/4)2^{\lfloor \log s \rfloor} \rfloor$ atoms, since $\{0, \dots, 2^{\lfloor \log s \rfloor}\} \in \mathcal{D}$. But this list is shorter than s , a contradiction to the incompressibility of \mathcal{F} .

We have proved that $T(s + n, s) > n \log s$. Since this holds for every n and s , we obtain an asymptotic lower bound of $\Omega(t \log s)$. \square

Actually, we obtained a lower bound on the time of a *retrieval* operation from a set of s data allowing for unbounded preprocessing time. In general, this is a stronger result than a lower bound on RASUS, which may be more difficult because all instructions must be simulated on-line. Also, it is easy to see that this lower bound holds also in the average where each of the s addresses is equally likely to be queried. Using this observation, we can also deduce that a randomized solution cannot have a better upper bound on expected running time.

5.3. AN APPLICATION: FINITE WORDS

THEOREM 4. *Let \mathcal{F}_k be the data type of k -bit words, with any primitive functions whatsoever. Then \mathcal{F}_k is incompressible.*

PROOF. Let \mathcal{D} be the set of all k -bit words; then $|\mathcal{D}| = 2^k$. Hence,


```

    for  $i = 0, \dots, s-1$ 
        storeR(pop(L), i)
    end for
advance:
    CopyImage( $R_1, S$ )
test:
    for  $j = 0, \dots, s-1$ 
        CopyImage( $S, R_1$ )
        loadR(j) /* sets  $t$  to the time it spends */
        if  $t > \log s$  then /* found a costly load */
            goto advance
        end if
    end for
    /* all the loads were cheap */
    S is the image sought.

```

FIGURE 3

$|\mathcal{G}^{n-1}| = 2^{k(n-1)} < 2^{kn} = |\mathcal{D}^{n-1}|$. Therefore, no function from \mathcal{D}^n to \mathcal{G}^{n-1} can be invertible. \square

COROLLARY 1. *Let \mathcal{T}_k be the data type of k -bit words, with any primitive instructions whatsoever. Then the \mathcal{T} -FLM requires $\Omega(t \log s)$ time to simulate the \mathcal{T} -RAM.*

In the next two sections, incompressibility proofs are given for other, more sophisticated, data types. The proofs apply new techniques to decision trees. The exact method depends on the data type at hand, but, in general, it can be summarized as follows: Discover a property of the primitive functions that prevents them from serving as "encoders" or "decoders." Exploiting the structure of decision trees, show that this property can be extended to a property of the finitely computable functions.

6. Simulation of the Real RAM

In this section the lower bound of Section 5 will be shown to apply to the simulation of the *real number* RAM. This model is the common abstraction used in algorithms dealing inherently with real numbers, such as in computational geometry and scientific computation. Shamos defines this model in [36] to have as *domain* the set of real numbers, and as primitive functions, the arithmetic operations $(+, -, \times, \div)$ and other common functions (such as exponents and logarithms, trigonometric functions, etc.), as needed, imposing only the restriction that all functions used be analytic (except for a finite set of points). Mild as it is, this restriction disallows some very useful functions such as the floor function (truncation to an integer). In fact, he [36] shows that one of his lower bounds does not hold with "floor" available. Our data type consists of the real numbers together with a set of primitives from the following set of functions.

Definition. By \mathcal{F} , we denote in this section the set of functions $f: \mathbb{R}^k \rightarrow \mathbb{R}^m$, for any $k, m > 0$, such that for some countable, closed set $C \subset \mathbb{R}^k$, f is continuous in $\mathbb{R}^k - C$.

\mathcal{F} includes all the functions mentioned above, analytic and not. Constants too may be seen as functions in this set, defined on \mathbb{R} , which are just independent of their argument. \mathcal{F} decision trees are similar to *algebraic*

decision trees [8], but the latter have a much more restricted primitive set (algebraic functions only).

By selecting \mathcal{F} as our primitive function set, we allow a model that is stronger than previous ones. The restriction on functions in \mathcal{F} is aimed at allowing functions whose continuity is perturbed, but in a limited way so that by composing any number of these functions, the perturbation may not grow too much. (Bijections between \mathbb{R}^n and \mathbb{R}^{n-1} will be more "perturbed" than we allow.)

Actually we show that under the above definition, the "perturbation" is limited to a special kind of set, a closed boundary set [24]. Recall that a *boundary set* is a subset A of a topological space such that A° (the interior of A) is empty (equivalently, the complement of A is a *dense set*). We denote the *complement* of A (with respect to the space under consideration) by \bar{A} , the *closure* of A by \bar{A} and the *boundary* of A by $\partial A = \bar{A} - A^\circ$. Note that ∂A is always closed, because \bar{A} is closed and A° is open. By a *neighborhood* (of a point) we mean any open set (containing the point). Thus, a boundary set is one that contains no neighborhood.

In this paper, we refer to \mathbb{R}^n as a topological space, actually the Euclidean n -space. This space is complete [24], which allows us to use the following theorem:

THEOREM 5. (BAIRE). *In a complete space, a countable union of closed boundary sets is a boundary set.*

Recall that \mathcal{F}^* denotes the closure of \mathcal{F} under function composition and aggregation.

THEOREM 6. *Let $k, m > 0$. Let $f: \mathbb{R}^k \rightarrow \mathbb{R}^m$ be computable by an \mathcal{F} -decision tree. Then there is an open set $O \subseteq \mathbb{R}^k$ such that f is continuous in O , and $\mathbb{R}^k - O$ is a boundary set.*

The set O displayed in the proof of this lemma will be called f 's *continuity domain*.

PROOF. We begin by proving the result for functions in \mathcal{F}^* , and this by induction on the number of compositions and aggregations used to construct f of functions from \mathcal{F} .

Basis: $f \in \mathcal{F}$. Then by definition, there is a countable, closed set $C \subset \mathbb{R}^k$ such that for $O = \mathbb{R}^k - C$, f is continuous in O . C is a boundary set, as every neighborhood in \mathbb{R}^k contains an uncountable number of points.

Induction step: Let us begin with aggregation. Assume $f: \mathbb{R}^k \rightarrow \mathbb{R}^n = (f_1, \dots, f_n)$ and assume our statement holds for all f_i . Let O_i be the continuity domain of f_i . Then f is clearly continuous in $O = \bigcap_{i=1}^n O_i$. Now $\mathbb{R}^k - O = \bar{O} = \bigcup_{i=1}^n \bar{O}_i$, a finite union of closed boundary sets. Thus, $\mathbb{R}^k - O$ is a closed boundary set.

Consider now $f = g(h)$ where $g \in \mathcal{F}$ and where h has a continuity domain $O \subseteq \mathbb{R}^k$. Let g be continuous in $\mathbb{R}^n - C$. Let $x \in O$; then h is continuous at x . If $h(x) \notin C$, then f is continuous at x . Also, if h is constant in some neighborhood of x , f is continuous at x . Therefore, the subset X of \mathbb{R}^k where

f is not continuous satisfies

$$X \subseteq \tilde{O} \cup \left(\bigcup_{y \in C} \partial(h^{-1}(\{y\})) \cap O \right). \quad (*)$$

Consider the restriction \hat{h} of h to O , where it is continuous. Note that $\partial(h^{-1}(\{y\})) \cap O = \partial(\hat{h}^{-1}(\{y\}))$. We show now that $Y = \bigcup_{y \in C} \partial(\hat{h}^{-1}(\{y\}))$ is closed in the subspace O . Let $x \in O$ be an accumulation point of Y , a limit of a sequence (x_i) , where $x_i \in \partial(\hat{h}^{-1}(\{y_i\}))$. By continuity of \hat{h} , we have $\hat{h}(x) = \lim_{i \rightarrow \infty} \hat{h}(x_i)$. For all i , $\hat{h}(x_i) \in C$, and since C is closed we have $y = \hat{h}(x) \in C$. Of course, $x \in \hat{h}^{-1}(\{y\})$. Suppose it is in its interior. Then there exists some k such that x_k belongs to this interior too (actually infinitely many elements do); $h(x_k)$ equals y ; therefore, x_k cannot belong to $\hat{h}^{-1}(\{y_i\})$ for any $y_i \neq y$. Thus, by the assumption on (x_i) , it must belong to $\partial \hat{h}^{-1}(\{y\})$, a contradiction. Therefore, $x \in \partial(\hat{h}^{-1}(\{y\})) \subseteq Y$.

Given that Y is closed in O , returning to the space \mathfrak{R}^k , we have

$$\bar{Y} \subseteq Y \cup \tilde{O}.$$

By the induction hypothesis, \tilde{O} is a boundary set. Assume that \bar{Y} contains a neighborhood N . Then, Y itself contains $N - \tilde{O}$, which is an open set as \tilde{O} is closed, and is not empty as, otherwise, N would be contained in \tilde{O} . So we have a neighborhood contained in $Y \cap O$. But this contradicts Baire's theorem since C being countable, Y is a countable union of closed boundary sets. Therefore, \bar{Y} is a boundary set. Returning to X , by (*) we have

$$X \subseteq \tilde{O} \cup \bar{Y},$$

which proves that \bar{X} is a boundary set. Moreover, f is continuous in $\mathfrak{R}^k - X \supseteq \mathfrak{R}^k - \bar{X}$, which is open, and hence fulfills our statement.

To extend the result to any function computable by an \mathcal{F} -decision tree, note that the following function on \mathfrak{R} belongs to \mathcal{F} :

$$t(x) \stackrel{\text{def}}{=} \begin{cases} 1 & x > 0, \\ 0 & x \leq 0. \end{cases}$$

It is easy to see that each part of an \mathcal{F} -decision tree consisting of a node ν , parent of two leaves λ_1 and λ_2 , can be replaced by a single leaf computing

$$t(f_\nu(x))f_{\lambda_1}(x) + (1 - t(f_\nu(x))f_{\lambda_1}(x)).$$

Repeating this process we can contract the whole tree to a single function in \mathcal{F}^* . \square

We now display the property of continuous mappings that actually prohibits compression. This property is preservation of topological dimension. We begin by reviewing the definition of this concept and some of its properties.

Definition [18, Sect. III.1]. The empty set and only the empty set has dimension -1 .

A topological space X has dimension $\leq n$ ($n \geq 0$) at a point p if p has arbitrarily small neighborhoods whose boundaries have dimension $\leq n - 1$.

X has dimension $\leq n$, $\dim X \leq n$, if X has dimension $\leq n$ at each of its points.

X has dimension n at a point p if it is true that X has dimension $\leq n$ at p , and it is false that X has dimension $\leq n - 1$ at p .

X has dimension n if $\dim X \leq n$ is true, and $\dim X \leq n - 1$ is false.

The following two theorems are also from [18].

THEOREM 7. *A subspace of a space of dimension $\leq n$ has dimension $\leq n$.*

THEOREM 8 (BROUWER). *Euclidean n -space has dimension n .*

It is obvious that the property of having dimension n is topologically invariant. Dimension is, however, not an invariant of continuous transformations: Peano's mapping of an interval on the whole of a square is an illustration of a continuous transformation that raises dimension, and projection of a plane into a line is an illustration of a transformation that lowers dimension. However, this transformation is not invertible, and thus does not interfere with incompressibility. Once the functions are required to be invertible, we obtain the property we need.

THEOREM 9. *Let $f: X \rightarrow Y$ be continuous and invertible, where X and Y are spaces such that $\dim Y = n$. Then, $\dim f^{-1}(Y) \leq n$.*

PROOF. Use induction on n .

Basis: $n = -1$.

$$\dim Y = -1 \Rightarrow Y = \emptyset \Rightarrow f^{-1}(Y) = \emptyset \Rightarrow \dim f^{-1}(Y) = -1.$$

Induction step: Assume the theorem holds for $n - 1$. Let p be an arbitrary point of $f^{-1}(Y)$ and U an arbitrary neighborhood of p therein; $f(U)$ is a subset of Y ; hence, by Theorem 7, $\dim f(U) \leq n$, where there exists, in the subspace $f(U)$, a neighborhood O of $f(p)$ such that $\dim \partial O \leq n - 1$. By f being continuous and invertible, it follows that $f^{-1}(O)$ is a neighborhood of p , contained in U , and also that

$$\partial f^{-1}(O) \subseteq f^{-1}(\partial O) \Rightarrow (\text{by the induction hypothesis and Theorem 7}),$$

$$\dim \partial f^{-1}(O) \leq n - 1.$$

Thus, $f^{-1}(O)$ is a neighborhood of p , arbitrarily small, and having boundary of dimension $\leq n - 1$; since p is an arbitrary point of $f^{-1}(Y)$, $\dim f^{-1}(Y) \leq n$, and the proof is complete. \square

THEOREM 10. *Let $\mathcal{T} = (\mathcal{R}, \mathcal{F}, <)$. Then \mathcal{T} is incompressible.*

PROOF. Assume by contradiction that there is a function $f: \mathcal{R}^n \rightarrow \mathcal{R}^m$ ($m < n$), finitely computable in \mathcal{T} , which is invertible. By Theorem 6, there exists a nonempty, open set O , where f is continuous; $f(O) \subseteq \mathcal{R}^m$; hence, by Theorems 7 and 8, we have $\dim f(O) \leq m$. By Lemma 9, also $\dim O \leq m$. But O contains a ball in \mathcal{R}^n , which clearly has dimension n ; hence, by Theorem 7, we have $\dim O \geq n$, a contradiction. \square

COROLLARY 2. *The real number FLM requires $\Omega(t \log s)$ time to simulate the real number RAM.*

These results carry over to the field of rational numbers. Here, we allow as primitives functions from \mathcal{F}_Q —the restriction of \mathcal{F} to functions that, when

applied to rational arguments, produce a rational result. We need a stronger version of Theorem 6, obtained by essentially the same proof. For the rest of this section, \mathcal{X} denotes a complete, dense in itself [24], subspace of \mathbb{R}^k .

THEOREM 6'. *Let $f: \mathcal{X} \rightarrow \mathbb{R}^m$ be computable by an \mathcal{F} decision tree. Then there is a set O , open in \mathcal{X} , such that f is continuous in O , and $\mathcal{X} - O$ is a boundary set in \mathcal{X} .*

THEOREM 11. *Let $\mathcal{F} = (Q, \mathcal{F}_Q, <)$. Then, \mathcal{F} is incompressible.*

PROOF. Assume, in contradiction, that there is an \mathcal{F}_Q decision tree computing a function f that maps Q^n injectively into Q^m , where $m < n$. Assume further that $g = f^{-1}$ is also finitely computable. Consider these functions as functions over the reals. As shown in Theorem 6, there exists a nonempty open set $O \subseteq \mathbb{R}^n$ in which f is continuous. $\overline{f(O)}$, as a subspace of \mathbb{R}^m , is complete and dense in itself. By Theorem 6', we have a nonempty set $N' \subseteq \overline{f(O)}$, open therein, in which g is continuous. The set $N = N' \cap f(O)$ cannot be empty and, relative to $f(O)$, is open. Thus, $f^{-1}(N)$ is open in O , hence in \mathbb{R}^n , and nonempty. We next show that f maps $f^{-1}(N)$ injectively onto N . This completes the proof because, as in the proof of Theorem 10, this cannot be true.

Let $x^1, x^2 \in f^{-1}(N)$ such that $f(x^1) = f(x^2) = y$; x^1 and x^2 may not both be rational; but anyway there are rational sequences $x_i^1 \rightarrow x^1$, $x_i^2 \rightarrow x^2$ in $f^{-1}(N)$. Let $y_i^j = f(x_i^j) \in N$. By continuity of f we have

$$y = f(x^1) = \lim_{i \rightarrow \infty} f(x_i^1) = \lim_{i \rightarrow \infty} y_i^1$$

entailing (by continuity of g)

$$g(y) = \lim_{i \rightarrow \infty} g(y_i^1) = \lim_{i \rightarrow \infty} x_i^1 = x^1.$$

Similarly, we get $g(y) = x^2$. Hence, $x^1 = x^2$. \square

COROLLARY 3. *The rational FLM requires $\Omega(t \log s)$ time to simulate the rational RAM.*

7. Simulation of the Integer RAM

The most common data type for the RAM is undoubtedly the integers. The domain of our data type is \mathcal{N} (the nonnegative integers). We define the function of *subtraction* on \mathcal{N} as producing 0 if the minuend is smaller than the subtrahend ('positive difference'). The choice of the *instruction set* for this data type is more subtle than for the previous two, since using a rather natural instruction set, integers may be compressed. In fact, there exists a solution to RASUS in the domain of integers which runs in $O(t\alpha(s))$ time, where the function $\alpha(s)$ is a (very slow growing) inverse of Ackermann's function [7]. In this simulation, only the operations *addition*, *subtraction*, and *shift* are used. The power of this algorithm stems apparently from the use of *shift*.

For the *shift* operation, we denote by $\text{shift}(x, y)$ the result of shifting x left by y bits, that is, $x \cdot 2^y$. We denote by $\text{shift}(x, -y)$ the result of shifting x right by y bits, that is, $\lfloor x/2^y \rfloor$.

The next theorem demonstrates that the right shift instruction is the source of the "breakthrough" in the above algorithm, by showing that excluding it from the instruction set leaves an incompressible data type.

7.1. AN INCOMPRESSIBILITY THEOREM FOR INTEGERS

THEOREM 12. *Let $\mathcal{F} = (\mathcal{A}, \mathcal{F}, <)$, where \mathcal{F} includes addition, subtraction, multiplication, any Boolean (bitwise) operations, integer division by two, and left shift. Then \mathcal{F} is incompressible.*

The property shared by all these functions, leading to the incompressibility result, is expressed by the following lemma.

LEMMA 4. *Let \mathcal{F} be the primitive function set of Theorem 12. Let $f: \mathcal{A}^n \rightarrow \mathcal{A}^m \in \mathcal{F}^*$, and $x, y \in \mathcal{A}^n$. Let h be the number of subexpressions of the form $\lfloor g/2 \rfloor$ that occur in f , and let $k > h$. Assume that for each subexpression of f of the form $(f_1 - f_2)$, or $\text{shift}(g_1, g_2)$, it is guaranteed that*

$$\begin{aligned} f_1(x) \geq f_2(x) & \quad f_1(y) \geq f_2(y), \\ g_2(x) < k & \quad g_2(y) < k. \end{aligned}$$

It then follows that

$$x \equiv y \pmod{2^k} \Rightarrow f(x) \equiv f(y) \pmod{2^{k-h}}.$$

PROOF. We prove that each of the primitives composing f , except $\lfloor g/2 \rfloor$, preserves congruence modulo 2^k . This is well known for addition, subtraction, and multiplication. Our use of *positive difference* does no harm because we are guaranteed that in evaluating $f(x)$ and $f(y)$, the operation either coincides with integer subtraction for both or produces zero for both. For the Boolean operations, the right k bits of the result are determined by the right k bits of the operands, and therefore they preserve congruence. The right k bits of $\text{shift}(a, b)$ are determined by the right k bits of a and by the value of b . We have two cases: Either $b < k$ for both x and y , which implies $g_2(x) = g_2(y)$, or $b \geq k$ for both, where the right k bits of both results are zero. Hence, the left shift too preserves congruence mod 2^k .

The lemma follows by structural induction on f .

Basis: $f \in \mathcal{F}$. There are two cases: (i) f is not $\lfloor x/2 \rfloor$. Then, as shown above, f preserves congruence modulo $2^k = 2^{k-h}$, and the lemma holds. (ii) $f(x) = \lfloor x/2 \rfloor$. Then, it is well known that

$$x \equiv y \pmod{2^k} \Rightarrow f(x) \equiv f(y) \pmod{2^{k-1}}.$$

Since $h = 1$, the lemma holds.

Induction step: A nonprimitive f may be constructed from simpler functions by means of *function composition* and *aggregation*. We first note that generally for $0 \leq i \leq j \leq k$, congruence modulo 2^{k-i} entails congruence modulo 2^{k-j} . We begin with aggregation. Let $f = (f_1, \dots, f_n)$ satisfy the lemma's assumptions. Then, they are clearly satisfied by each of the f_i . For $i = 1, \dots, n$, let h_i be the number of subexpressions of the form $\lfloor g/2 \rfloor$

occurring in f_i . Then, the number of such occurrences in f is $h = \sum_{i=1}^n h_i$. In particular, $h \geq h_i$ for all i . Hence, by the induction hypothesis, each f_i preserves congruence modulo 2^{k-h_i} and therefore modulo 2^{k-h} . Put together, f preserves congruence modulo 2^{k-h} .

Consider now $f = g(\phi)$. Fulfillment of the lemma's assumptions by f , x , and y means that they are satisfied by ϕ , x , and y and by g , $\phi(x)$, and $\phi(y)$. Let h_1 be the number of occurrences of division by two in ϕ and h_2 the number of such occurrences in g . By the induction hypothesis,

$$\begin{aligned} x \equiv y \pmod{2^k} &\Rightarrow \phi(x) \equiv \phi(y) \pmod{2^{k-h_1}} \\ &\Rightarrow g(\phi(x)) \equiv g(\phi(y)) \pmod{2^{k-h_1-h_2}}. \end{aligned}$$

For f the number of occurrences is $h = h_1 + h_2$, and hence we get $f(x) \equiv f(y) \pmod{2^{k-h}}$. \square

We are now ready to prove the theorem.

Assume by contradiction that there is a decision tree in our model that computes a surjective function from \mathcal{A}^m to \mathcal{A}^n where $n > m$ (this is required for a "decoding algorithm"). Let t be the number of leaves in this tree. For each leaf λ , let R_λ be the set of n -tuples produced at this leaf. Then

$$\bigcup_{\lambda} R_\lambda = \mathcal{A}^n.$$

Let h be the maximal number of occurrences of division by two in the functions of the leaves. Choose $k > \log t + h$. For each leaf λ , denote by X_λ the set of m -tuples for which the tree delivers the result at λ . Then, $R_\lambda = f_\lambda(X_\lambda)$, with $f_\lambda \in \mathcal{F}^*$. We can assume that for all pairs $x, y \in X_\lambda$ the assumptions of Lemma 4 hold with respect to f_λ , since otherwise we can enforce them by replacing λ with a finite subtree whose nodes test all the relations mentioned in the lemma such that for each leaf only one outcome of these tests is possible.

Hence, it follows from the lemma that

$$\begin{aligned} |R_\lambda \bmod 2^{k-h}| &= |f_\lambda(X_\lambda) \bmod 2^{k-h}| \leq |X_\lambda \bmod 2^k| \\ &\leq |\mathcal{A}^m \bmod 2^k| = 2^{km}. \end{aligned}$$

Therefore

$$\begin{aligned} |\mathcal{A}^n \bmod 2^k| &= \left| \bigcup_{\lambda} R_\lambda \bmod 2^k \right| \\ &\leq 2^h \left| \bigcup_{\lambda} R_\lambda \bmod 2^{k-h} \right| \\ &\leq 2^h \sum_{\lambda} |R_\lambda \bmod 2^{k-h}| \\ &\leq 2^h \cdot t \cdot 2^{km} < 2^k \cdot 2^{km} = 2^{k(m+1)} \leq 2^{kn} \end{aligned}$$

while actually, $|\mathcal{A}^n \bmod 2^k| = 2^{kn}$ — a contradiction. \square

COROLLARY 4. *The integer FLM (with the primitives of Theorem 12) requires $\Omega(t \log s)$ time to simulate the integer RAM.*

Remark. Mathematically, *left shift* is a special case of *exponentiation* a^b , which could have been entered into our primitive set as such; and we would still have incompressibility, although the proof gets a little harder. Exponentiation has a more arithmetic flavor than left shift, and correspondingly, we would add *integer division* instead of adding *right shift* in our next theorem. This would allow to implement the right shift as $\lfloor x/2^j \rfloor$.

7.2. A COMPRESSION SCHEME USING SHIFTS. Let the binary representation of x be $b_n \dots b_1 b_0$. We denote by $\text{Right}_d(x)$ the right d bits of x , that is, the number $b_{d-1} \dots b_1 b_0$. Right_d is computable by

$$\text{Right}_d(x) = x - \text{shift}(\text{shift}(x, -d), d).$$

By $\text{Mid}_{i,d}(x)$, where $i, d \geq 0$, we denote the number $b_{i+d-1} \dots b_i$, computed by

$$\text{Mid}_{i,d}(x) = \text{Right}_d(\text{shift}(x, -i)).$$

Our compression function encodes n integers, for any fixed n , in just a pair of numbers. The compressed structure is called *a packet*.

A packet P represents an n -tuple of integers $P[0], \dots, P[n-1]$. We write $P = P^{(n)}$ for emphasis. The actual representation of a packet is as the pair (a, b) where for all $0 \leq i < n$, $P[i] \in \{0, \dots, 2^b - 1\}$ and $a = \sum_{i=0}^{n-1} \text{shift}(P[i], ib)$.

Building a packet, given the n numbers, is straightforward; we are free to choose b at our convenience; for example, as the maximum of $P[0], \dots, P[n-1]$. In order to do without multiplication, we can use for the "block size" of the packet not b but 2^b , exploiting the fact that $i \cdot 2^b = \text{shift}(i, b)$. The packet would then be defined by $a = \sum_{i=0}^{n-1} \text{shift}(P[i], i \cdot 2^b)$, and multiplication would not be required.

Each of the packet members can be retrieved in constant time, using the following relation (using the form with multiplication):

$$P[i] = \text{Mid}_{ib, b}(a).$$

This allows for a decompression function of $O(n)$ running time. We omit a detailed verification that is straightforward. It is worth mentioning, however, how this compression scheme can be employed by a fast RASUS. Clearly, by encoding all the used portion of the RAM's memory in a packet, a constant time *load* is easily achieved. However, for each store we may have to recreate the packet because of the new datum being larger than the block size. To obtain a fast *store*, we have to use a more complex structure. A solution for this problem is given in [7], where packet hierarchies are maintained to obtain a RASUS of $O(\alpha(s))$ time per operation, where $\alpha(s)$ is a functional inverse of Ackermann's function. This function grows slower than every inverse of a primitive recursive function. The computational model used in [7] is weaker than the PLM; it has for memory just a finite number of registers.

8. Historical Notes

In the introduction, we described a machine model used by Tarjan [38, 39] that resembles the LISP machine. However, its usage in this paper was very specific, not using the power of high-level data types. Thus, it was closer, as

Tarjan states, to a machine model previously suggested by Knuth [19]. His model, called a *linking automaton*, has the same structure as Tarjan's pointer machine (similar to the FLM), but was constrained to operate on a finite alphabet of symbols.

This machine belongs to the class of *combinatory* or *discrete* machines, such as Turing's machine, which arose from the theory of computability, and are still dominant in the more fundamental areas of complexity theory. These machines operate on a finite alphabet, or even less: a similar model, called a *storage modification machine* (SMM), has been independently proposed by Schönhage [33–35]. This machine operates on a memory that does not store any kind of "data"; the information is comprised in the linking pattern of the nodes. The SMM is easily proved to be real-time equivalent to the linking automaton. Schönhage's comprehensive paper [35] defines the model, shows that the SMM can simulate a multidimensional Turing machine in real time, and proves Schnorr's observation that they are real-time equivalent to the so-called successor RAMs [32]. The *successor RAM* (SRAM) is a machine with a memory addressable by \mathcal{A} , but an input/output alphabet of $\{0, 1\}$ and an internal data type of integers including only the successor function and comparison for equality. Thus, this machine falls short of our "high-level" assumptions (Section 1), and this explains its weakness: It has such a feeble data type that instead of addressing memory with its "data," you can do as well with pointers.

We should also note the first appearance of an abstract computing machine that operates on a graph structure, proposed by Kolmogorov and Uspenskii [21, 23]. Their model was intended to give a flexible and realistic definition of an algorithm. It is different from the above by having its memory graph *undirected*, with a bounded-node degree. The "programming" of the machine is also different, since it is mathematically oriented and not similar to actual programming as in the other machines. Kolmogorov–Uspenskii machines (KUMs) can be simulated in real time by SMMs, but whether this is possible in the inverse direction is not yet known.²

Following Tarjan's paper, several authors have used the pointer machine as a model for studying data structure problems. Blum [10] complements Tarjan's paper in studying the disjoint set union problem. Harel and Tarjan [15] demonstrate a lower bound of $\Omega(m \log \log n)$ for performing m *nearest-common-ancestor* queries on a pointer machine. In contrast, an $O(m)$ algorithm is given for a RAM. They present this problem as a case for comparing the pointer machine with the RAM. However, their result does not shed light on our problem, since their formulation of the problem for the pointer machine lower bound differs from that for the RAM; while the RAM represents tree nodes as integers, the pointer machine is to represent them as unique nodes.

A similar result appears in [25], where Mehlhorn et al. shows tight bounds for some variants of the union-split-find problem. One of the variants, which can be solved by a RAM in $O(1)$ steps per operation, requires $O(\log \log n)$ pointer machine steps per operation or $O(\log n)$ steps if set separability is assumed (a further restriction on the representation of sets in the pointer machine).

² This appears as Problem 8 of [14].

Chazelle [11] also gives a lower bound for a pointer machine. He addresses *orthogonal range searching* in d dimensions (where all the s answers to the query are to be reported). On the pointer machine, a query time of $O(s + \text{polylog}(n))$ time can only be achieved at the expense of $\Omega(n(\log n / \log \log n)^{d-1})$ space, which is optimal. On the contrary, as he showed in [12], a RAM can process a query in time $O(s + \log n)$ using $O(n \log^\epsilon n)$ storage, for any $\epsilon > 0$. Thus, in his case, the pointer machine requires asymptotically more storage to achieve the same query time. However, here again the problem is presented to the pointer model in a different way.

Paige [26] presents considerations similar to ours with respect to the choice of models and defines a high-level RAM as the machine model of an *array-processing language* and a pointer machine as modeling a *list-based language*. This resulted in models that are practically equivalent to our \mathcal{F} -RAM and \mathcal{F} -LM. He even discusses the simulation of a RAM by the list machine, and complements the results of this paper by describing a large class of practical algorithms that can be simulated by an LM without asymptotic overhead.

9. Conclusion

We have defined the *incompressibility* property for a general data type. Using this property, we have proved an $O(t \log s)$ tight bound on the time required by a pointer machine to simulate a RAM. Our models of computation are defined in a general way for a high-level data type, extending the traditional definition of such models. This may be the first time that a complexity lower bound is proved in such a general and powerful model.

Actually, we presented the RAM and LISP machines as representatives of a class of high-level models, the \mathcal{F} -machines. These make an attractive setting for the design and analysis of algorithms because, in fact, algorithm designers have already assumed such flexibility. Incompressibility proved to be a tool strong enough for proving lower bounds in models of this kind.

We have proved the incompressibility property for the data types of finite words, integers, and real numbers with a generous set of primitives. The primitive sets are large enough to be useful in practical, high-level algorithms and are perhaps the strongest for which lower bounds have been achieved. For the integers, we exclude the right-shift operation, which is shown to induce compressibility.

The proof of the lower bound followed the following scheme: We first proved a lower bound in a model whose power is restricted by assuming the data to be *symbols*. The proof generalized to more complex data types in a natural way by replacing the restriction of symbols by the property of incompressibility. This scheme may carry on to other problems where a lower bound is known assuming symbols are used, or where other similar assumptions have been made, as in Aggarwal and Vitter's lower bounds on I/O complexity [2], who call their restrictive assumption *indivisibility of the records* and pose the task of relaxing this assumption as an open problem.

The family of high-level models includes further variations on the theme of memory access. The *Hierarchical Memory Model* (abbreviated HMM), introduced by Aggarwal et al. [1], operates exactly as the RAM, but memory access instructions are charged according to the address used. Although all other instructions cost one, an access to memory location a costs proportion-

ally to log a (or to another cost function). The virtues of this model are discussed in [1], and we do not repeat them here. A natural extension of our study is to compare this model to the LM just as we did for the RAM. This problem is addressed in [4]. It is shown that the HMM relates to the LM much as the LM relates to the RAM. The proofs of this relationship resemble those of the previous chapters, providing further evidence to the adequacy of incompressibility arguments to this kind of problems.

NOTE ADDED IN PROOF. It was remarked in Section 7 that some of the published complexity results attributed to "pointer machines" are incomparable with results for the RAM, because the pointer machine was not given the same problem. To make this distinction clear, we suggest to adopt the term, *pointer algorithms*, for procedures that are given a linked structure and whose task is to modify it in some manner. Such procedures can be used as subroutines in larger programs, but have no independent value. The results in [11], [15], and [25] pertain to pointer algorithms and are not applicable to solutions of the stand-alone problem where the input is given in raw form, say as numbers. It is the latter type of problem that was solved by the RAM algorithms of [12], [15], and [25]. On the other hand, among the papers referenced, independent algorithms for *pointer machines*, allowing for comparison of machine models, only appear in Paige's work [26] and in our papers.

ACKNOWLEDGMENTS. The authors wish to thank Michael Tarsi for suggesting the use of Brouwer's Theorem in Section 6. Also the comments of an anonymous referee helped much in "debugging" the paper.

REFERENCES

1. AGGARWAL, A., ALPERN, B., CHANDRA, A. K., AND SNIR, M. A model for hierarchical memory. In *Proceedings of the 19th ACM Symposium on Theory of Computing*. ACM, New York, 1987, pp. 305-314.
2. AGGARWAL, A., AND VITTER, J. S. The I/O complexity of sorting and related problems. In *Proceedings of the 14th ICALP*. 1987, pp. 467-478.
3. AHO, A. V., HOPCROFT, J. E., AND ULLMAN, J. D. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, Reading, Mass. 1974.
4. BEN-AMRAM, A. M. On addresses versus pointers. Tech. Rep. 173/90. Tel Aviv Univ., Tel Aviv, Israel.
5. BEN-AMRAM, A. M. On pointers versus addresses. MS dissertation. Dept. Comput. Sci., Tel-Aviv University, Tel-Aviv, Israel, 1988.
6. BEN-AMRAM, A. M., AND GALIL, Z. On pointers versus addresses. In *Proceedings of the 29th Annual IEEE Symposium on Foundations of Computer Science* (White Plains, N.Y.). IEEE, New York, 1988, pp. 532-538.
7. BEN-AMRAM, A. M., AND GALIL, Z. On the power of shifting integers. *Inf. Comput.*, to appear.
8. BEN-OR, M. Lower bounds on algebraic computation trees. In *Proceedings of the 15th Annual ACM Symposium on Theory of Computing*. ACM, New York, 1983, pp. 80-86.
9. BLUM, L., SHUB, M., AND SMALE, S. On a theory of computation over the real numbers; NP completeness, recursive functions and universal machines. In *Proceedings of the 29th Annual IEEE Symposium on Foundations of Computer Science* (White Plains, N.Y.). IEEE, New York, 1988.
10. BLUM, N. On the single-operation worst-case time complexity of the disjoint set union problem. *SIAM J. Comput.* 15, 4 (1986), 1021-1024.
11. CHAZELLE, B. Lower bounds on the complexity of multidimensional searching. In *Proceedings of the 27th Annual IEEE Symposium on Foundations of Computer Science*. IEEE, New York, 1986, pp. 87-96.
12. CHAZELLE, B. A functional approach to data structures and its use in multidimensional searching. Tech. Rep. TR CS-85-16, Brown Univ., 1985.
13. COOK, S. A., AND RECKHOW, R. A. Time bounded random access machines. *J. Comput. Syst. Sci.* 7, 4 (1973), 354-375.
14. DURIS, P., GALIL, Z., PAUL, W., AND REISCHUK, R. Two nonlinear lower bounds for on-line computations. *Inf. Control* 60 (1984), 1-11.

15. HAREL, D., AND TARJAN, R. E. Fast algorithms for finding nearest common ancestors. *SIAM J. Comput.* 13 (1984), 338-355.
16. HART, S., AND SHARIR, M. Nonlinearity of Davenport-Schinzel sequences and of a generalized path compression scheme. In *Proceedings of the 25th Annual IEEE Symposium on Foundations of Computer Science* (Singer Island, Fla.). IEEE, New York, 1984, pp. 313-319.
17. HARTMANIS, J., AND SIMON, J. On the power of multiplication in random access machines. In *Proceedings of the 15th Annual Symposium on Switching and Automata Theory* (New Orleans, La.). IEEE, New York, 1974, pp. 13-23.
18. HUREWICZ, W., AND WALLMAN, H. *Dimension Theory*. Princeton University Press, Princeton, N.J., 1948.
19. KNUTH, D. E. *The Art of Computer Programming*, vol. 1. *Fundamental Algorithms*. Addison-Wesley, Reading, Mass., 1968.
20. KNUTH, D. E. *The Art of Computer Programming*, vol. 3. *Sorting and Searching*. Addison-Wesley, Reading, Mass., 1975.
21. KOLMOGOROV, A. N. On the notion of algorithm. *Uspehi Mat. Nauk.* 3 (1953), 175-176.
22. KOLMOGOROV, A. N. Three approaches to the definition of the concept of "amount of information." *Problemy Peredachi Informacii* 1, 1 (1965), 3-11. English translation in *Selected Transl. in Math. Statistics and Probability*, 7 (1965), 293-302.
23. KOLMOGOROV, A. N., AND USPENSKII, V. A. On the definition of an algorithm. *Uspehi Mat. Nauk.* 13 (1958), 3-28. English translation in *AMS Transl. II* 29 (1963), 217-245.
24. KURATOWSKI, K. *Introduction to Set Theory and Topology* [BORON, L. F. (transl.)]. Addison-Wesley, Reading, Mass., 1962.
25. MEHLHORN, K., NÄHER, S., AND ALT, H. A lower bound for the complexity of the union-split-find problem. In *Proceedings of the 14th ICALP* (Karlsruhe). Springer-Verlag, New York, 1987, pp. 479-488.
26. PAIGE, R. Real-time simulation of a set machine on a RAM. In W. Koczkodaj, ed., *International Conference on Computing and Information (ICCI '89)* (Toronto, Ont., Canada). *Comput. Inf.* 2, (1989), 69-73.
27. PAUL, W. J. Kolmogorov complexity and lower bounds. In Lothar Budach, ed., *Fundamentals of Computation Theory (FCT '79)*, Akademie-Verlag, Berlin, 1979, pp. 325-334.
28. PAUL, W. J., SEIFERAS, J. I., AND SIMON, J. An information-theoretic approach to time bounds for on-line computation. *J. Comput. Syst. Sci.* 23, 2 (1981), 108-126.
29. PRATT, V. R., RABIN, M. O., AND STOCKMEYER, L. J. A characterization of the power of vector machines. In *Proceedings of the 6th Annual ACM Symposium on Theory of Computing* (Seattle, Wash., Apr. 30-May 2). ACM, New York, 1974, pp. 122-134.
30. PREPARATA, F. P., AND SHAMOS, M. I. *Computational Geometry: An Introduction*. Springer-Verlag, New York, 1985.
31. SARNAK, N., AND TARJAN, R. E. Planar point location using persistent search trees. *Commun. ACM* 29, 7 (July 1986), 669-679.
32. SCHNORR, C. P. *Rekursive Funktionen und ihre Komplexität*. Teubner, Stuttgart, 1974.
33. SCHÖNHAGE, A. Real-time simulation of multi-dimensional Turing Machines by storage modification machines. In *Project MAC Technical Memorandum*, vol. 37. MIT, Cambridge, Mass., 1973.
34. SCHÖNHAGE, A. Storage modification machines. In *Theoretical Computer Science-4th GI Conference Aachen*, 1979.
35. SCHÖNHAGE, A. Storage modification machines. *SIAM J. Comput.* 9, 3 (1980), 490-508.
36. SHAMOS, M. I. *Computational Geometry*, Ph.D. dissertation. Yale, Cambridge, Mass., 1978.
37. SIKLÓSSY, L. *Let's Talk LISP*. Prentice-Hall, Englewood Cliffs, N.J., 1976.
38. TARJAN, R. E. Reference machines require nonlinear time to maintain disjoint sets. In *Proceedings of the 9th Annual ACM Symposium on Theory of Computing* (Boulder, Col., May 2-4). ACM, New York, 1977, pp. 18-29.
39. TARJAN, R. E. A class of algorithms which require nonlinear time to maintain disjoint sets. *J. Comput. Syst. Sci.* 18 (1979), 110-127.

RECEIVED MARCH 1989; REVISED MARCH AND OCTOBER 1990; ACCEPTED OCTOBER 1990

μ Database: Parallelism in a Memory-Mapped Environment (Research Summary)

Peter A. Buhr, Anil K. Goel, Naomi Nishimura, and Prabhakar Ragde

Department of Computer Science, University of Waterloo, Waterloo, Ontario, Canada N2L 3G1
{pabuhr,akgoel,nishi,plragde}@uwaterloo.ca

Abstract

The goal of the μ Database project is to understand the behaviour of data structures and their algorithms, both parallel and sequential, in a memory-mapped environment. Memory mapping allows primary-memory pointer-based data structures to be stored on secondary storage, and subsequently traversed and modified, without transforming the embedded pointers. The project incorporates a collaboration between practitioners and theoreticians and has produced a toolkit to support a parallel memory-mapped environment, extensive concurrency tools, an analytical model for the system validated by experiments, and preliminary results involving parallel database join algorithms as well as sequential B-tree and R-tree data structures. Future work includes augmenting the toolkit, developing and testing more parallel algorithms tuned for efficiency in a memory-mapped environment, and extending the current model to cover more general algorithms and capture more low-level details of the physical machine.

1 Introduction

Successful parallel environments must take advantage of not only process parallelism (multiple CPUs) but also data parallelism (data partitioned across multiple disks). Data parallelism is particularly important; currently it is the most effective mechanism to deal with the increasing disparity between CPU and I/O performance. Traditional techniques for building and managing data structures on secondary storage involve constructing complex, often application-specific, buffer management subsystems using explicit I/O calls. To facilitate code reuse and rapid prototyping and to decrease programming complexity, I/O mechanisms should be made simpler and more transparent, particularly in parallel environments.

Manipulating persistent data structures presents further difficulties due to the incompatibility between addresses in primary memory (pointers) and secondary storage (disk block addresses); pointers can not normally be stored directly on secondary storage for further usage, and creating and manipulating complex structures without pointers

is both cumbersome and expensive. One approach is to extend primary storage practices and tools to apply to secondary storage, thereby creating a *single-level store*, which eliminates the need for expensive execution-time conversions of structured data between primary and secondary storage, while allowing the power of a general purpose programming language to be extended to secondary storage. The goal is to achieve substantial performance advantages over conventional file access [6, 7, 13, 23, 16, 20] in both the development and execution of algorithms.

Of particular interest is the implementation of a single-level store by memory-mapping, that is, the use of virtual memory to map data stored on secondary storage into primary storage so that the data is directly accessible by the processors' instructions [30, 11, 29]. Instead of accessing data on disk by explicit read and write routine calls, all I/O is performed implicitly by the operating system during normal address dereference through the virtual memory hardware. Deferring I/O until data is actually referenced and taking advantage of available memory-management hardware significantly reduces complexity of persistent data structures while decreasing the runtime cost.

2 μ Database Project

We are developing μ Database [11, 9], a C++-based toolkit for building persistent data structures using a single-level store implemented by memory mapping. Our single-level store and others, such as Objectstore [16], Texas [25], and QuickStore [29], are differentiated by their approaches to the *address consistency* problem. This problem results when a pointer-based data structure built at a particular location in memory can not subsequently be reloaded at that location because it is otherwise occupied. A commonly used solution is *pointer swizzling*, which consists of placing the data structure at a new location in memory and modifying its pointers to reflect the new location of data. Swizzling requires the ability to locate all embedded pointers and modify them, which adds to the runtime cost. The performance loss is especially significant for operations that incur high overhead in data preparation, such as sequential scans, where the data is accessed only once, and operations that implicitly fetch and prepare data multiple times in large data structures using small primary storage. As well, applications that access multiple structures simultaneously cause a large number of address collisions resulting in more pointer swizzling.

μ Database adopts a different solution to the address consistency problem, called the *exact positioning of data ap*

Permission to make digital/hard copies of all or part of this material for personal or classroom use is granted without fee provided that the copies are not made or distributed for profit or commercial advantage, the copyright notice, the title of the publication and its date appear, and notice is given that copyright is by permission of the ACM, Inc. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires specific permission and/or fee.

SPAA '96, Padua, Italy

© 1996 ACM 0-89791-809-6/96/06 ..\$3.50

Proceedings of the
ACM Symposium on Parallel Algorithms and Architecture¹⁹⁹⁶

pp. 196-199

June 1996

proach. We employ a novel technique that allows application of an old solution, hardware segmentation, to the problem: each hardware segment is an address space starting at a virtual zero, in which a persistent data structure can be built, stored, and subsequently retrieved and modified. Multiple segments can be simultaneously accessed in a single application because each segment has its own non-conflicting address space. When a segment is mapped into memory, pointers *within* the segment do not require modification; pointers *outside* the segment may require modification, but in general, these pointers represent a small percentage of the total number of pointers in a data structure.

2.1 Concurrency Tools

We have developed extensive concurrency tools for use in building highly concurrent data structures, involving both processor and data parallelism, and have found that a single-level store allows many existing concurrent algorithms and data structures to be used directly for persistent data. Basic concurrency capabilities are provided in a C++-based toolkit, called μ C++ [8, 12]. μ Database uses μ C++ to allow a programmer to build whatever form of concurrency is appropriate. Using a spectrum of μ C++ tools, processor parallelism may be specified at a low level, where a lock is used to protect data, or at a high level, where a light-weight server task controls access to data. As well, the granularity of data access may be individual fields, entire objects, or groups of objects. In general, processor parallelism, e.g., concurrent access to a B-tree, involves trade-offs between maximizing concurrency and the administrative overhead of fine-grain locking. While achieving near-optimal concurrency is often data-structure specific, we believe it is possible to provide general concurrency tools that are applicable to broad classes of data structures to simplify application development. Some of these general concurrency tools are provided as part of the μ Database toolkit.

Data parallelism deals with the CPU-I/O bottleneck by partitioning data across multiple disks and then accessing the data in parallel [19, 28]. The transition between the two toolkits occurs at this point with μ Database providing the ability to partition a single-level store transparently across multiple disks with μ C++ providing parallelism even while I/O operations are occurring. As for processor parallelism, we have developed general tools, which support a variety of partitioning schemes and parallel algorithms, that form part of the μ Database toolkit.

2.2 Algorithm Design

The novel persistence facility of μ Database and concurrency capabilities of μ C++ yield new degrees of freedom for designing complex and highly-parallel database algorithms: traditional algorithms must be re-examined in this new framework. For example, we have been able to improve performance of algorithms by making use of spatial ordering information implicit in the pointers defining data structure relations. As well, there is a need for partitioning schemes [22] to distribute data across multiple disks to achieve maximum parallelism; performance is a function of the dynamic and possibly concurrent access pattern, which is often independent of the static structure of the data.

A natural starting point for the study of database algorithms is the join operation as it is "...relevant to relational,

extensible, and object-oriented database systems alike" [14]. Joining is a merging of data from two collections of data objects, R and S , where an R object contains a join attribute that refers to an S object, and data from each is combined to form the join. In pointer-based join algorithms [24, 17, 10], the join attribute is a virtual pointer to objects in S . We have designed and analyzed parallel pointer-based versions of nested loops, sort-merge, and Grace join algorithms. As stated above, it is possible to exploit the implicit ordering of objects in S via the virtual pointers to eliminate the usual sorting or hashing of S in sort-merge and hash-based joins, respectively.

2.3 Analytical Modelling

We have developed an analytical model of our memory-mapped environment, providing quantitative predictions of performance. The model acts as a high-level filter to predict general performance behaviour; only those algorithms that look promising from the model need to be more fully tested. More importantly, a quantitative model is an essential tool for subsystems such as a database query optimizer which use static analysis to construct a fast implementation.

Since prediction of actual running time rather than characterizing asymptotic complexity was the goal of our model, our work is closer to models developed to explain empirical results [24, 21, 18] than extensions to theoretical models handling such features as I/O complexity, block transfer, and hierarchical memory [2, 1, 4, 15, 3, 26, 27]. As modelling the physical machine closely and accurately is essential, we have, for example, used a measured function to model disk transfer times, to account for all of the following factors. In a persistent memory-mapped environment, all I/O is performed by means of normal virtual memory mechanisms. Thus, a read page fault needs to be processed before execution can continue, whereas write-backs can be handled in parallel with program execution. Further, I/O costs are affected by whether data on the disk(s) are accessed sequentially or randomly. The total span of disk over which random accesses take place is also a factor in determining the amortized cost of I/O. Full details of the model can be found in [10].

2.4 Practical Relevance

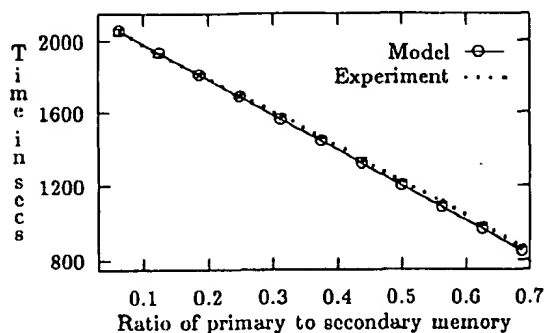
Besides developing a theoretical framework for studying memory-mapped systems, the μ Database project addresses several important practical issues. Some of these issues are reduction in programming complexity while maintaining or improving execution speed, accurate cost estimation, exploitation of data and process parallelism with memory-mapping to address the increasing CPU-I/O disparity, and effective tools for teaching concurrency and parallelism in persistent systems.

3 Preliminary Results and Future work

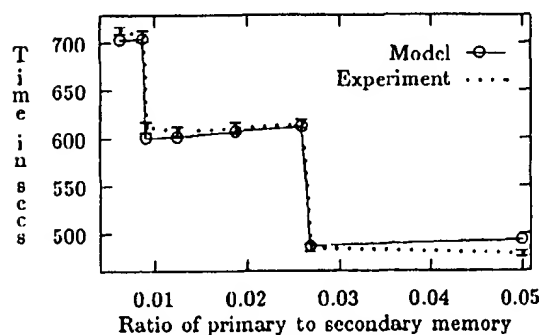
To validate our model and analysis, experiments were run that performed full joins on two relations R and S with 102,400 objects each [10]. R and S were partitioned across 4 disks with one R and one S partition on each disk. All experiments were run on a Sequent Symmetry with 10 processors, which uses a simple page replacement algorithm (see [11] for testbed details). The execution environment was strictly

controlled so the results were not influenced by any outside activity. The amount of memory for the experiment's address space and the global cache were tightly controlled as well.

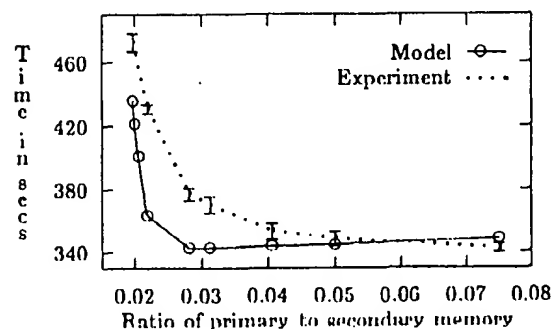
Fig. 1 shows the predicted and measured elapsed times for running the various join algorithms with varying amounts of available memory. The discontinuities in the sort-merge graph occur when additional merging phases are required. The curve in the Grace graph at low memory levels results from thrashing caused by the page replacement algorithm. As is evident from the graphs, our model does an excel-



(a) Nested Loops



(b) Sort Merge



(c) Grace

Figure 1: Experimental Results

lent job of predicting performance for the various join algorithms in almost all conditions. In particular, there is a close match between prediction and actual performance for nested loops and sort-merge. For Grace, our approximation for I/O caused by thrashing at low memory levels is reason-

ably accurate; there is scope for further refinement of this approximation. A major part of the difference between prediction and actual behavior at low memory levels comes from the overhead introduced by the particular page replacement strategy used by the operating system.

Future studies with the model include speedup and scaleup measurements, changing the nature of the joining relations and a comparative analysis of various algorithms. Our analysis of the join algorithms highlights an inherent drawback in memory-mapped single-level stores: the lack of control over buffer management on the part of the database application results in incorrect decisions being made at times by the underlying page replacement strategy. While accepting this inefficiency, we demonstrated two approaches for achieving predictable behaviour, an essential property in a database system. With single-level stores becoming more common, it is our hope that future research and development in operating system architecture will make it feasible for database applications to exercise more control over the replacement strategies used (see [5]). There is also scope for further improvement in the design of our model, especially in the modelling of the underlying paging behaviour. Future work will involve extending our model to other memory-mapped environments, allowing us to perform comparative studies. It will also be an interesting exercise to explore the applicability of our model in traditional environments.

References

- [1] Aggarwal, A., Alpern, B., Chandra, A. K., and Snir, M. "A Model for Hierarchical Memory". In *ACM STOC*, pages 305-314, May 1987.
- [2] Aggarwal, A., Chandra, A. K., and Snir, M. "Communication Complexity of PRAMs". *Theoretical Comput. Sci.* 71(1):3-28, Jan. 1990.
- [3] Aggarwal, A. and Vitter, J. S. "The Input/Output Complexity of Sorting and Related Problems". *Commun. ACM*, 31(9):1116-1127, Sept. 1988.
- [4] Alpern, B., Carter, L., Feig, E., and Selker, T. "Uniform Memory Hierarchies". *Algorithmica*, 12(2/3):72-109, August/September 1994.
- [5] Appel, A. W. and Li, K. "Virtual Memory Primitives for User Programs". In *4th International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 96-107, Apr. 1991.
- [6] Atkinson, M. P., Bailey, P. J., Chisholm, K. J., Cockshott, P. W., and Morrison, R. "An Approach to Persistent Programming". *The Computer Journal*, 26(4):360-365, Nov. 1983.
- [7] Atkinson, M. P. and Morrison, R. "Procedures as Persistent Data Objects". *ACM Trans. Prog. Lang. Syst.*, 7(4):539-559, Oct. 1985.
- [8] Buhr, P. A., Ditchfield, G., Strooboscher, R. A., Younger, B. M., and Zarnke, C. R. "μC++: Concurrency in the Object-Oriented Language C++". *Software—Practice and Experience*, 22(2):137-172, Feb. 1992.

- [9] Buhr, P. A. and Goel, A. K. "µDatabase Annotated Reference Manual, Version 1.0". Technical Report Unnumbered: Available via ftp from plg.uwaterloo.ca in pub/uDatabase/uDatabase.ps.gz, Department of Computer Science, University of Waterloo, Waterloo, Ontario, Canada, N2L 3G1, June 1995.
- [10] Buhr, P. A., Goel, A. K., Nishimura, N., and Ragde, P. "Parallel Pointer-Based Join Algorithms in Memory Mapped Environments". To appear in the *Proceedings of the 12th IEEE International Conference on Data Engineering*, Feb. 1996.
- [11] Buhr, P. A., Goel, A. K., and Wai, A. "µDatabase : A Toolkit for Constructing Memory Mapped Databases". In Albano, A. and Morrison, R., editors, *Persistent Object Systems*, pages 166-185, Sept. 1992. Springer-Verlag.
- [12] Buhr, P. A. and Strooboscher, R. A. "µC++ Annotated Reference Manual, Version 4.4". Technical Report Unnumbered: Available via ftp from plg.uwaterloo.ca in pub/uSystem/uC++.ps.gz, Department of Computer Science, University of Waterloo, Waterloo, Ontario, Canada, N2L 3G1, Sept. 1995.
- [13] Copeland, G., Franklin, M., and Weikum, G. "Uniform Object Management". In *Advances in Database Technology - Proc. European Conference on Database Technology*, pages 253-268, Mar. 1990.
- [14] Graefe, G. "Sort-Merge-Join: An Idea whose Time has(h) Passed?". In *IEEE International Conference on Data Engineering*, page 406, February 1994.
- [15] Hong, J.-W. and Kung, H. T. "I/O Complexity: The Red-Blue Pebble Game". In *ACM STOC*, pages 326-333, 1981.
- [16] Lamb, C., Landis, G., Orenstein, J., and Weinreb, D. "The Objectstore Database System". *Commun. ACM*, 34(10):50-63, Oct. 1991.
- [17] Lieuwen, D. F., DeWitt, D. J., and Mehta, M. "Pointer-Based Join Techniques for Object-Oriented Databases". In *International Conference on Parallel and Distributed Information Systems*, Jan. 1993.
- [18] Martin, T. P., Larson, P.-A., and Deshpande, V. "Parallel Hash-Based Join Algorithms for a Shared-Everything Environment". *IEEE Transactions on Knowledge and Data Engineering*, 6(5):750-763, Oct. 1994.
- [19] Patterson, D. A., Gibson, G., and Katz, R. H. "A Case for Redundant Arrays of Inexpensive Disks(RAID)". In *ACM SIGMOD*, pages 109-116, June 1988.
- [20] Richardson, J. E., Carey, M. J., and Schuh, D. T. "The Design of the E Programming Language". *ACM Trans. Prog. Lang. Syst.*, 15(3):494-534, July 1993.
- [21] Schneider, D. A. and DeWitt, D. J. "A Performance Evaluation of Four Parallel Joins Algorithms in a Shared-Nothing Multiprocessor Environment". In *ACM SIGMOD*, pages 110-121, June 1989.
- [22] Seeger, B. and Larson, P.-A. "Multi-Disk B-trees". In *ACM SIGMOD*, pages 436-445, June 1991.
- [23] Shekita, E. and Zwilling, M. "Cricket: A Mapped, Persistent Object Store". In Dearle, A. et al., editors, *Implementing Persistent Object Bases: Principles and Practice*, Proceedings of the Fourth International Workshop on Persistent Object Systems", pages 89-102. Morgan Kaufmann, 1990.
- [24] Shekita, E. J. and Carey, M. J. "A Performance Evaluation of Pointer-Based Joins". In *ACM SIGMOD*, pages 300-311, June 1990.
- [25] Singhal, V., Kakkad, S. V., and Wilson, P. R. "Texas: An Efficient, Portable Persistent Store". In Albano, A. and Morrison, R., editors, *Persistent Object Systems*, pages 11-33, Sept. 1992. Springer-Verlag.
- [26] Vitter, J. S. and Shriver, E. A. M. "Algorithms for Parallel Memory, I: Two-Level Memories". *Algorithmica*, 12(2/3):110-147, August/September 1994.
- [27] Vitter, J. S. and Shriver, E. A. M. "Algorithms for Parallel Memory, II: Hierarchical Multi-Level Memories". *Algorithmica*, 12(2/3):148-169, August/September 1994.
- [28] Weikum, G., Zäbbak, P., and Scheuermann, P. "Dynamic File Allocation in Disk Arrays". In *ACM SIGMOD*, pages 406-415, June 1991.
- [29] White, S. J. and DeWitt, D. J. "QuickStore: A High Performance Mapped Object Store". In *ACM SIGMOD*, pages 395-406, May 1994.
- [30] Wilson, P. R. "Pointer Swizzling at Page Fault Time: Efficiently Supporting Huge Address Spaces on Standard Hardware". *Computer Architecture News*, 19(4):6-13, June 1991.

**NOTICE OF OFFICE PLAN TO CEASE SUPPLYING COPIES OF CITED U.S. PATENT
REFERENCES WITH OFFICE ACTIONS, AND PILOT TO EVALUATE THE
ALTERNATIVE OF PROVIDING ELECTRONIC ACCESS TO SUCH U.S. PATENT
REFERENCES**

Summary

The United States Patent and Trademark Office (Office or USPTO) plans in the near future to: (1) cease mailing copies of U.S. patents and U.S. patent application publications (US patent references) with Office actions except for citations made during the international stage of an international application under the Patent Cooperation Treaty and those made during reexamination proceedings; and (2) provide electronic access to, with convenient downloading capability of, the US patent references cited in an Office action via the Office's private Patent Application Information Retrieval (PAIR) system which has a new feature called "E-Patent Reference." Before ceasing to provide copies of U.S. patent references with Office actions, the Office shall test the feasibility of the E-Patent Reference feature by conducting a two-month pilot project starting with Office actions mailed after December 1, 2003. The Office shall evaluate the pilot project and publish the results in a notice which will be posted on the Office's web site (www.USPTO.gov) and in the Patent Official Gazette (O.G.). In order to use the new E-Patent Reference feature during the pilot period, or when the Office ceases to send copies of U.S. patent references with Office actions, the applicant must: (1) obtain a digital certificate from the Office; (2) obtain a customer number from the Office, and (3) properly associate applications with the customer number. The pilot project does not involve or affect the current Office practice of supplying paper copies of foreign patent documents and non-patent literature with Office actions. Paper copies of references will continue to be provided by the USPTO for searches and written opinions prepared by the USPTO for international applications during the international stage and for reexamination proceedings.

Description of Pilot Project to Provide Electronic Access to Cited U.S. Patent References

On December 1, 2003, the Office will make available a new feature, E-Patent Reference, in the Office's private PAIR system, to allow more convenient downloading of U.S. patents and U.S. patent application publications. The new feature will allow an authorized user of private PAIR to download some or all of the U.S. patents and U.S. patent application publications cited by an examiner on form PTO-892 in Office actions, as well as U.S. patents and U.S. patent application publications submitted by applicants on form PTO/SB08 (1449) as part of an IDS. The retrieval of some or all of the documents may be performed in one downloading step with the documents encoded as Adobe Portable Document format (.pdf) files, which is an improvement over the current page-by-page retrieval capability from other USPTO systems.

Steps to Use the New E-Patent Reference Feature During the Pilot Project and Thereafter

Access to private PAIR is required to utilize E-Patent Reference. If you don't already have access to private PAIR, the Office urges practitioners, and applicants not represented by a practitioner, to take advantage of the transition period to obtain a no-cost USPTO Public Key Infrastructure (PKI) digital certificate, obtain a USPTO customer number, associate all of their pending and new application filings with their customer number, install no-cost software (supplied by the Office) required to access private PAIR and E-Patent Reference feature, and make appropriate arrangements for Internet access. The full instructions for obtaining a PKI digital certificate are available at the Office's Electronic Business Center (EBC) web page at: <http://www.uspto.gov/ebc/downloads.html>. Note that a notarized signature will be required to obtain a digital certificate.

To get a Customer Number, download and complete the Customer Number Request form, PTO-SB125, at: <http://www.uspto.gov/web/forms/sb0125.pdf>. The completed form can then be transmitted by facsimile to the Electronic Business Center at (703) 308-2840, or mailed to the address on the form. If you are a registered attorney or patent agent, then your registration number must be associated with your customer number. This is accomplished by adding your registration number to the Customer Number Request form. A description of associating a customer number with an application is described at the EBC web page at: http://www.uspto.gov/ebc/registration_pair.html.

The E-Patent Reference feature will be accessed using a new button on the private PAIR screen. Ordinarily all of the cited U.S. patent and U.S. patent application publication references will be available over the Internet using the Office's new E-Patent Reference feature. The size of the references to be downloaded will be displayed by E-Patent Reference so the download time can be estimated. Applicants and registered practitioners can select to download all of the references or any combination of cited references. Selected references will be downloaded as complete documents as Adobe Portable Document Format (.pdf) files. For a limited period of time, the USPTO will include a copy of this notice with Office actions to encourage applicants to use this new feature and, if needed, to take the steps outlined above in order to be able to utilize this new feature during the pilot and thereafter.

During the two-month pilot, the Office will evaluate the stability and capacity of the E-Patent Reference feature to reliably provide electronic access to cited U.S. patent and U.S. patent application publication references. While copies of U.S. patent and U.S. patent application publication references cited by examiners will continue to be mailed with Office actions during the pilot project, applicants are encouraged to use the private PAIR and the E-Patent Reference feature to electronically access and download cited U.S. patent and U.S. patent application publication references so the Office will be able to objectively evaluate its performance. The public is encouraged to submit comments to the Office on the usability and performance of the E-Patent Reference feature during the pilot. Further, during the pilot period registered practitioners, and applicants not represented by a practitioner, are encouraged to experiment with the feature, develop a proficiency in using the feature, and establish new internal processes for using the new access to the cited U.S. patents and U.S. patent application publications to prepare for the anticipated cessation of the current Office practice of supplying copies of such cited

references. The Office plans to continue to provide access to the E-Patent Reference feature during its evaluation of the pilot.

Comments

Comments concerning the E-Patent Reference feature should be in writing and directed to the Electronic Business Center (EBC) at the USPTO by electronic mail at eReference@uspto.gov or by facsimile to (703) 308-2840. Comments will be posted and made available for public inspection. To ensure that comments are considered in the evaluation of the pilot project, comments should be submitted in writing by January 15, 2004.

Comments with respect to specific applications should be sent to the Technology Centers' customer service centers. Comments concerning digital certificates, customer numbers, and associating customer numbers with applications should be sent to the Electronic Business Center (EBC) at the USPTO by facsimile at (703) 308-2840 or by e-mail at EBC@uspto.gov.

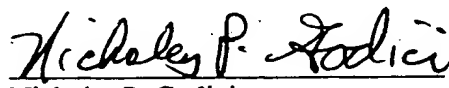
Implementation after Pilot

After the pilot, its evaluation, and publication of a subsequent notice as indicated above, the Office expects to implement its plan to cease mailing paper copies of U.S. patent references cited during examination of non provisional applications on or after February 2, 2004; although copies of cited foreign patent documents, as well as non-patent literature, will still be mailed to the applicant until such time as substantially all applications have been scanned into IFW.

For Further Information Contact

Technical information on the operation of the IFW system can be found on the USPTO website at <http://www.uspto.gov/web/patents/ifw/index.html>. Comments concerning the E-Patent Reference feature and questions concerning the operation of the PAIR system should be directed to the EBC at the USPTO at (866) 217-9197. The EBC may also be contacted by facsimile at (703) 308-2840 or by e-mail at EBC@uspto.gov.

Date. 12/1/03


Nicholas P. Godici
Commissioner for Patents